TOWARDS BUILDING RELIABLE SOLAR-POWERED REMOTELY-DEPLOYED SENSING SYSTEMS

BY

YONG YANG

DISsertation

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Doctoral Committee:

Professor Tarek F. Abdelzaher, Chair & Director of Research
Professor Jiawei Han
Associate Professor Indranil Gupta
Dr. Jie Liu, Microsoft Research
Abstract

Driven by new demands in both civil and national-security applications such as environmental monitoring and border control, sensing devices will not only be deployed at home and in urban areas, but also pervade every corner of the world. Remotely deployed systems have to work in an unattended manner, face harsh and complex environments, and rely on unstable energy sources (e.g., solar energy). Therefore, building remote sensing systems and collect data of interests from them are confronted with unique challenges.

First, data collection is subject to loss in communication because wireless links that such systems rely on are error-prone by nature. Second, because of the limited connectivity to the outside world via wireless communication, the sensory data have to be stored in the system when the remote deployment is disconnected from the basestation, and thus are facing uncontrolled loss in storage caused by physical dynamics. Third, when remotely-deployed nodes become unresponsive, it is generally hard to determine what caused the anomalous silence and assess the status of the data collection process without sending a person to the field. Furthermore, the dynamic nature of the energy source calls for new system designs.

In this thesis, we present a suite of our work on addressing the above challenges. In particular, we propose adaptive schemes to dynamically adjust the coding redundancy used to mitigate the data loss in communication and storage under time varying energy constraints, and we propose a tele-diagnostic tool to automatically infer node states based on its power consumption traces. The proposed work has been evaluated on a real solar-powered sensing testbed that we designed and deployed.
To my parents, for their love and support.
Acknowledgments

Foremost, I am sincerely thankful to my thesis advisor, Professor Tarek F. Abdelzher for his support and guidance on my Ph.D. study and thesis. In many ways I have learnt much from him, and this thesis would not have been possible without him.

My great gratitude goes to Professor Jiawei Han, Professor Indranil Gupta and Dr. Jie Liu for serving on my doctoral committee and for their insightful comments on this thesis. In addition, I would like to thank Professor P.R. Kumar for helping me in several optimization problems related to the thesis.

I want to express my special thank to my former advisor, late Professor Jennifer C. Hou. With her passion and immense knowledge, she had enlighten me in my early Ph.D. years, and enhanced my interests in research. I will remember her for my life.

I would like to extend my gratitude to the fellow labmates in the UIUC Cyber Physical Computing Group, especially Lu Su, Maifi Khan, Hieu Le, Dong Noh and Lili Wang, for their great help in our collaborated projects and papers. Also, I would like to thank my coauthors, including but not limited to Professor Carl Gunter, Dr. Nageswara S. Rao, Dr. Mallikarjn Shanka, Professor David Yau, Jren-Chit Chin, Yan Gao, I-Hong Hou, Lu-Chuan Kung, Michael LeMay, Chris Ma and Parya Moinzadeh.

I thank NSF for supporting my attendance to the prestigious conference INFOCOM, MOBISYS and MOBICOM, which have inspired me in many aspects of my research. I also thank the Vodafone U.S. foundation for supporting my doctoral study.

Lastly, and most importantly, I wish to thank my parents, Gangben Yang and Fengge Du, for their love and support. To them I dedicate this thesis.
# Table of Contents

List of Tables ........................................................................................................ vii
List of Figures .......................................................................................................... viii

Chapter 1  Introduction .......................................................................................... 1
  1.1 Motivation and Research Challenges ............................................................... 1
  1.2 Contributions .................................................................................................. 4
  1.3 Thesis Organization ......................................................................................... 7

Chapter 2  A Solar-powered Sensing Testbed ...................................................... 8
  2.1 Outdoor Testbed .............................................................................................. 8
    2.1.1 Energy Subsystem ...................................................................................... 9
    2.1.2 Computing Subsystem .............................................................................. 10
  2.2 Indoor Testbed ................................................................................................ 13

Chapter 3  Utilizing Erasure Codes for Reliable Data Delivery ...................... 15
  3.1 Introduction .................................................................................................... 15
  3.2 System Model and Problem Formulation ......................................................... 18
    3.2.1 Network Model ........................................................................................ 18
    3.2.2 Erasure Coding Model .......................................................................... 19
    3.2.3 Energy Model .......................................................................................... 20
    3.2.4 Problem Formulation ............................................................................. 21
    3.2.5 Discussion ................................................................................................ 23
  3.3 SolarCode: an Effective Approximated Solution ........................................ 23
    3.3.1 A Close Lower Bound of the Objective .................................................. 24
    3.3.2 Concavity of $Pr(\alpha)$ ......................................................................... 26
    3.3.3 Centralized Solution ............................................................................... 28
    3.3.4 Decentralized Solution .......................................................................... 29
  3.4 Implementation Issues .................................................................................... 33
  3.5 Performance Evaluation ................................................................................. 34
    3.5.1 SolarCode with Projected Solar Trace ..................................................... 37
    3.5.2 SolarCode with Real Solar Trace ............................................................... 39
    3.5.3 An Extensive Experiment ....................................................................... 39
    3.5.4 Comparison to Other Schemes ................................................................. 40
  3.6 Conclusion ...................................................................................................... 41
# Chapter 4 Reliable In-network Data Storage

4.1 Introduction .................................................. 43  
4.2 SolarStore: A Threshold-based Storage Replication Service .......... 46  
  4.2.1 Problem Formulation ..................................... 46  
  4.2.2 A Dynamic Threshold-based Solution ...................... 48  
  4.2.3 Data Encoding and Dissemination Design .................. 51  
  4.2.4 Implementation Issues .................................. 52  
  4.2.5 Performance Evaluation ................................ 55  
4.3 SolarQoS: Providing Quality of Storage based on Static Utility .... 62  
  4.3.1 Problem Formulation ..................................... 62  
  4.3.2 Solution .................................................. 69  
  4.3.3 Performance Evaluation ................................ 70  
4.4 SimStore: A Content-aware Storage Service based on Dynamic Similarity 76  
  4.4.1 Problem Formulation ..................................... 78  
  4.4.2 Local Storage Replacement Policy ......................... 80  
  4.4.3 Distributed Storage Replacement Protocol .................. 82  
  4.4.4 Performance Evaluation ................................ 87  
4.5 Conclusion .................................................. 97

# Chapter 5 Power-based Diagnosis of Node Silence

5.1 Introduction .................................................. 98  
5.2 General Design of Tele-Diagnostic Subsystem ....................... 101  
5.3 Implementation on Our Testbed ................................ 104  
  5.3.1 Hardware .................................................. 104  
  5.3.2 Diagnosis Objectives .................................... 106  
5.4 Powertracer: Diagnostics Based on Passive Sampling ............... 108  
5.5 Power Watermarking: Diagnostics based on Active Signature Generation 118  
  5.5.1 Watermark Generator .................................... 118  
  5.5.2 Watermark Detection .................................... 122  
  5.5.3 Evaluation ............................................... 124  
5.6 Conclusion .................................................. 130

# Chapter 6 Related Work

6.1 Solar-powered Remotely-deployed Sensing Systems ................ 132  
6.2 Reliable Data Delivery in Sensing Systems ........................ 134  
6.3 In-network Data Storage in Sensing Systems ...................... 135  
6.4 Remote System Diagnosis .................................... 137

# Chapter 7 Conclusions and Future Work

References .................................................... 143
List of Tables

3.1 Packet delivery probability for links used by at least one flow. .......... 35
3.2 Data rate and route of 6 flows used in the evaluation. ............... 35
3.3 Average end-to-end packet delivery probability of different schemes in each of
the two scenarios. .......................................................... 41

4.1 Node Average Reliability Level. .............................................. 59
4.2 Data collection rates (byte/s) of the four applications on the 9 nodes. ... 71
4.3 Average replication factors ($\alpha$) of the four applications by SolarQoS with the
predicted or real solar energy trace. The last column is the overall utility
achieved. ............................................................................ 74
4.4 The utility ratio of SolarQoS over NoRep and NoQoS under different utility
assignments. ................................................................. 75
4.5 The number of events and associated audio clips used in the experiments. 89

5.1 Confusion matrix for MM with clustering (Window size=30 minutes, Number
of clusters=50, Normalization used: Outlier filtering) ..................... 117
List of Figures

1.1 Research Overview .............................................. 5
2.1 Map of 9 deployed nodes. ....................................... 9
2.2 Outside look of one node. ...................................... 9
2.3 Components inside of a node enclosure. ..................... 10
2.4 Basic components of each node. ............................... 11
2.5 Each node in the indoor testbed has the same computing subsystem as nodes outdoors, while the energy subsystem is emulated by using current sensors, X10 modules and controller, and real solar energy traces collected outdoor. 13

3.1 A simple topology in which two flows $a$ and $b$, originated from node $n_1$ and $n_2$ respectively, go to node $n_4$ through a relay node $n_3$. The residual energy of $n_3$ is shown on the right. ........................................ 16
3.2 With $b = 8$, $Pr(\alpha)$ has an inflection point if $p_m = 0.01$, but is concave if $p_m = 0.1$ or 0.43. .......................................................... 26
3.3 The lower bound for $p_m$ to guarantee the concavity of $Pr(\alpha)$. .................. 26
3.4 Real and projected solar energy traces for one day. ............. 34
3.5 Node topology. .................................................. 34
3.6 Real (thin red curve) and projected (thick blue curve) solar energy traces during Oct 21st - Nov 4th 2008. Total energy collected daily in units of Ampere Hour is also shown: the numbers on the first line are for the real trace and the second are for the projected trace. .................. 35
3.7 The replication factor $\alpha$ (top), solar energy input (middle) and residual energy (bottom) for the experiment with flow $f_1$ and $f_2$ for 5 days. .................. 36
3.8 The residual energy of node 1, 2, 5 and 7 when SolarCode is not used. ............ 38
3.9 The replication factor $\alpha$ by SolarCode with the real energy trace as an oracle. 39
3.10 The replication factor $\alpha$ by SolarCode for the experiment of all 6 flows for 15 days. Only 5 links are shown for the clarity of presentation. .................. 40
3.11 The replication factor $\alpha$ by FullCode, which uses erasure codes only when the battery is full. ............................................. 41

4.1 One node drowned in the flood by heavy rains. .................. 44
4.2 Architecture of SolarStore. ..................................... 53
4.3 Residual energy $E_{\text{residual}}$ and the threshold $\frac{P_{\text{sys}}}{P_{\text{solar}}}C$. .................. 56
4.4 Changes of the residual storage space $S_{\text{residual}}$ and storage surplus $\Delta S$. .... 58
4.5 Average reliability level $\tilde{\rho}$ of the data on node 9. ........................................ 59
4.6 Residual energy $E_{\text{residual}}$ and the threshold $\frac{P_{\text{solar}}}{P_{\text{solar}}} C$ of node 9 under a highly skewed solar energy input. ................................................................. 60
4.7 Changes of the residual storage space $S_{\text{residual}}$ and storage surplus $\Delta S$ of node 9 under a highly skewed solar energy input. ................................................................. 61
4.8 The total data loss, caused by blackout and node failure, in the case that 1, 2 or 3 nodes failed. ....................................................................................................................... 62
4.9 The lower bound for the number of sensor nodes $N$ to guarantee the concavity of $Pr(\alpha, 1)$. ................................................................. 69
4.10 The replication factor $\alpha$ of node 8 (top) and node 9 (middle) for 7 days. The bottom part is the predicted solar energy trace. ................................. 72
4.11 The residual energy in the batteries of node 8 for the two cases when SolarQoS is used and not used. ................................................................. 73
4.12 Comparison of SolarQos, NoQoS and NoRep. ................................................................. 75
4.13 Example of integrating the local clustering ($k=1$) of two nodes into one global clustering ($k=1+1=2$). ................................................................. 83
4.14 Overview of SimStore. ......................... 83
4.15 The examples of appearance and call waveform of the three bird species. 88
4.16 An illustration of some data samples (i.e., audio clips) in a two-dimensional feature space formed by the total power and signal centroid. The feature values are normalized by z-scores. ................................................................. 90
4.17 CDFs of distances between similar data (inner cluster) and dissimilar data (inter cluster) under different distance functions. ................................. 92
4.18 Scheme comparison by BCubed scores. ................................................................. 93
4.19 Scheme comparison by event coverage ratio. ................................................................. 95
4.20 Message overhead of Global-DA and SimStore. ................................................................. 95
4.21 Energy overhead of Global-DA and SimStore. ................................................................. 96

5.1 A power-based tele-diagnostic system as an “add-on” to a sensing system. The watermark generator is needed only when active watermarking is used. 102
5.2 Our power meter with a wireless radio. ................................................................. 104
5.3 An indoor node with a power meter measuring its power consumption. ................................................................. 105
5.4 Possible failure states in our system that cause a node to become unresponsive. App-I is the application responsible for sensing sound using microphone. App-II is the application responsible for recording images. ................................................................. 107
5.5 Power traces of a sensor node in three states: (a) router fails and application is sampling sound, (b) antenna fails and application crashes, and (c) OS crash. ................................................................. 109
5.6 Effect of number of MM states on classification accuracy (Window size=30 minutes, Data preprocessing used: Outlier filtering) ................................................................. 111
5.7 Effect of sampling rate on the classification accuracy of MM (Window size=30 minutes, Number of states=50, Data preprocessing used: Outlier filtering) ................................................................. 112
5.8 Effect of window size on the classification accuracy of MM (Number of states=50, Data preprocessing used: Outlier filtering) ................................................................. 113
5.9 Effect of window size on the classification accuracy of MM based on z-score
(Number of states=50, Data preprocessing used: Outlier filtering, z-score
normalization) ................................................................. 114
5.10 Effect of window size on the classification accuracy of MM based on signal
difference (Number of states=50, Data preprocessing used: Outlier filtering,
difference between consecutive signal) ................................ 115
5.11 Effect of window size on the classification accuracy of MM with clustering
(Number of clusters=50, Data preprocessing used: Outlier filtering) ..... 116
5.12 The CPU usage of the sleep function, an application and their aggregation. 121
5.13 Similarity scores and detection thresholds for three cases. ..................... 125
5.14 Effect of window size on accuracy. .................................................. 126
5.15 Effect of sampling rate on accuracy. .................................................. 126
5.16 Effect of heartbeat period on accuracy. ............................................. 127
5.17 Effect of number of applications on accuracy. .................................... 128
5.18 Effect of CPU load on accuracy. ...................................................... 129
5.19 Energy efficiency of heartbeat generation. ......................................... 129
Chapter 1

Introduction

1.1 Motivation and Research Challenges

Advances in MEMS sensors, embedded processing and ad-hoc wireless networks have placed Cyber-physical Systems (CPS) [1, 2, 3] on stage as the next computing revolution. The cyber space and the physical space become closely coupled in CPS through integrating sensing, computing, communication and storage capabilities efficiently and dependably. In particular, networked sensing devices embedded within physical elements provide critical foundations for cyber-physical systems to acquire data about the physical world.

In line with the proliferation of cyber-physical systems, we envision that sensing devices will not only be deployed at home and in urban areas, but also pervade every corner of the world. For example, various sensors have been built into normal items like cloth, appliances, cars, and buildings to support human-centric applications and create smart living environments [4, 5, 6, 7, 8]. Meanwhile, precision agriculture [9, 10] utilizes sensors deployed in fields to collect soil and crop data to enhance the efficiency and growth of cultivations. Environmental monitoring uses deployments of unattended sensors in remote areas to collect data for environmental or animal habitat studies [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]. In addition, sensing instruments are also deployed to remotely detect intrusions and track targets [21, 22], as well as to remotely watch the healthiness of civil infrastructures such as highways [23] and pipelines [24].

Urban deployments and remote deployments of sensing devices share many common research problems, such as topology control [25, 26, 27], traffic routing [28, 29, 30] and load
balancing [31, 32, 33]. However, remote deployments are facing unique challenges because of their characteristics. Remote deployments are usually outdoor, facing a much more complex and harsher environments. Remote deployments are left unattended for most of the time, and the cost of in-situ system maintenance is high. Also, applications on such systems typically require long-running (or even perpetual) operations, which makes solar energy one of the most suitable energy sources. However, the dynamic nature of solar energy introduces new challenges into the system design.

The primary usage of sensing system deployments is to collect data of interest (e.g. sound, motion, temperature, and other variables) from physical environments and deliver them safely to end users. Therefore, in this thesis, we focus on the research problems in building reliable remotely-deployed solar-powered sensing systems, with special emphasis on how to make the data collection process more reliable and thus maximize the sensory data that can be retrieved from the deployment. Specifically, we address the following research challenges.

[Reliable End-to-end Data Delivery]: Remotely deployed nodes typically rely on wireless communication to deliver sensory data to basestations. Because of the error-prone nature of wireless communication, a series of solutions has been proposed in the literature on reliable data delivery in wireless sensor networks. However, most of the prior work has focused on traditional battery-powered nodes, while the renewable and dynamic nature of solar energy brings new opportunities and challenges to this problem. Reliability can be improved by using redundancy. However, achieving redundancy takes additional energy. Thus, in traditional battery-powered systems, exploiting any level of redundancy will reduce the network lifetime because of the finite amount of energy in node batteries. However, solar-powered sensor nodes have incentive to spend extra energy, especially when the battery is fully charged, because this energy surplus would be wasted otherwise. Thus, by adjusting the energy spending in accordance with the energy harvesting, more energy could be harvested and hence more work could be accomplished. Therefore, a research challenge is how to
adjust the extra energy spending on redundancy so that the total amount of data delivered is maximized while the network lifetime is conserved.

[**Reliable In-network Data Storage**]: In many scenarios, instead of having continuous connectivity to the basestation, sensor nodes deployed in remote locations may only have very limited or even no connectivity to the outside world. Consequently, the sensory data have to be stored in the network until the next uploading opportunity appears (e.g., periodically collected by data mules or uploaded through satellites). Hardware failures could occur to nodes placed in the outdoor harsh environments because of all sorts of natural events (such as fire or flood). Thus, the stored data are subject to loss because of hardware failures. Storage reliability can also be achieved by using redundancy. However, the extra energy spending on storage reliability may lead to more energy depletion, which suspends the sensing process and results in more data loss. Therefore, this calls for an adaptive storage scheme that dynamically adjusts the redundancy level of the cached data (protect them from loss due to node failures) according to time-varying energy constraints. Furthermore, different sensory data may have different criticality/utility to end users. For instance, in audio sensing applications, audio data with sound are more interesting than those with just background noise. Considering the limited energy and storage resources on sensor nodes, it is desirable to allow data with higher criticality/utility to end users to have higher priority in using the energy and storage resources to achieve higher overall utility.

[**Remote Diagnosis of Node Failures**]: When remotely-deployed nodes become unresponsive, it is generally hard to determine what caused some node to become silent, without sending a person to the field. If the cost of such field trips is high, remote diagnosis becomes highly desirable to assess the urgency of human intervention, which may depend on the cause of silence. For example, if the cause of the problem is energy depletion, there may not be much that can be done about it until the energy source is restored (e.g., weather improves). If the sensing applications on an unresponsive node are still running and collecting data (albeit unable to communicate for some reason), immediate human intervention may not be
strictly required. However, if a failure has affected the whole node causing data collection to stop, then early intervention is preferable in order to minimize data loss. Prior work has proposed a set of tools to troubleshoot a deployed node by collect its runtime states via regular radio. However, we have to rely on other side channels to infer node states since silent nodes, by definition, cannot communicate via its main radio.

In fact, other problems also exist in building solar-powered remotely-deployed sensing systems. However, we will focus on the above ones in this thesis because they are the major challenges from a data-centric point of view, which regards sensory data as the most valuable output of sensing systems. Moreover, the above challenges are motivated by the real needs originated from a real solar-powered sensing testbed that we have built in a field to the south of the University of Illinois at Urbana-Champaign campus. This testbed has been running since August 2008 to record bird vocalizations and nesting activities. The collected data are being used by scientists in the department of natural resources and environmental sciences for bird habitat studies.

Actually, the above problems have been discussed at length in previous liberation, but mainly in the context of low-end sensor nodes (e.g., Tmote [34] and MicaZ [35] motes) powered by traditional batteries. In this thesis, however, we focus on high-end sensing systems. Because of the limited capability of low-end sensor nodes on sensing, computation and communication, high-end sensing systems have to be employed in many application scenarios for high-bandwidth data acquisition. Their special properties (e.g., relatively high cost and high capacity) may make solutions that are infeasible for low-end systems become feasible, which necessities the investigation on such particular systems.

1.2 Contributions

In this thesis, we present a suite of our work to address the above challenges in building solar-powered remotely-deployed sensing systems. As shown in Figure 1.1, our work pro-
vides supports for sensing applications in three different layers, making the data collection process more reliable and maximizing the amount of interesting data delivered to end users. Specifically, our contributions are listed as follows.

- **To improve data communication reliability**, at the network layer, we proposed a protocol, called **SolarCode** [36], that utilizes erasure coding to recover data from partial packet loss or corruption. It adaptively adjusts the level of coding redundancy according to the availability of solar energy, so that the total amount of data delivered is maximized while the network lifetime is conserved. We formulate the problem as an optimization problem, which is in general hard to solve because of the special curvature of its objective function. By exploiting special properties of the problem, we proposed an effective approximate solution that has a constant approximation ratio.

- **To improve data in-network storage reliability**, we proposed a storage service, called **SolarStore** [11], that adaptively generate data replications by using erasure codes according to the availability of solar energy. Although we can formulate this as an optimization problem as in SolarCode, solving this optimization requires knowledge of both the node failure probability distribution (to predict replication needs) and
weather patterns (to predict energy input). Thus, we devised a simple but effective solution that is independent of these two physical models.

- To accommodate data with different criticality/utility levels to end users, we proposed an enhanced storage scheme, called SolarQoS. We formulated an optimization problem with the objective to maximize the total utility of the retrievable data, where a numeric value is used to represent the static utility of each type of sensory data.

- Furthermore, instead of being static, data utility could also be dynamically changing. Therefore, we proposed a content-aware storage scheme, called SimStore, that dynamically measures the utility of data points based on its similarity with other data. When a new data point arrives, it decides which data point should be replaced so that the diversity of the stored data is maximized.

- To diagnose an unresponsive node, inspired by the fact that sensing, computation and storage take power to be properly carried out, we proposed Powertracer [37], that uses consumed power traces as a side channel to infer system states. Powertracer works as an add-on to the original deployed system. It includes a low-cost power meter, one per sensor node, that periodically samples the current and voltage of its host node. These meters are wirelessly connected via direct, low-bandwidth links to a low-end basestation, where Powertracer uses a pre-trained classifier to determine current system state.

- Powertracer requires prior training for each possible failure state. As the number of applications increases, the number of possible states grows exponentially, making them increasingly more difficult to classify. Although it is applicable in many deployments that have very specific purposes and thus do not typically run a wide range of different applications concurrently, we proposed another diagnostic scheme, called Power Watermarking [38], to further improve the diagnostic performance under multiple
application scenarios. It places a module into each host to actively inject unique power patterns (watermarks) into the power consumption traces based on the current system status. Since watermarks adhere to a pre-agreed-upon code, there is no need for prior training.

- We have designed and deployed a solar-powered sensing testbed, which has been used by researchers in both the department of computer science and the department of natural resources and environmental sciences to facilitate their research. It has been running since August 2008, and is one of the longest-running testbeds in the research community.

1.3 Thesis Organization

This thesis is organized as follows. This chapter provides the motivation of the research and the overview of the research contributions. Before going into the details of the proposed work, we first present our solar-power testbed in Chapter 2 as it is the testbed on which we implemented and tested our proposed schemes. Chapter 3 introduces the communication protocol SolarCode to improve the end-to-end data delivery reliability. Chapter 4 presents a series of our solutions to address the challenge in reliable in-network storage when the deployed system is disconnected from the outside world. Chapter 5 presents our power-based diagnostic tool powertracer with its enhancement power watermarking. Chapter 6 surveys the related work. Chapter 7 concludes this thesis and discusses future research.
Chapter 2

A Solar-powered Sensing Testbed

We have designed and set up a solar-powered sensing system testbed. In addition, we have built an indoor testbed with the purpose of performance evaluation under a wide range of emulated environmental conditions. Since these are the testbeds on which we implemented and tested our proposed schemes, we first describe them in this chapter before presenting the detailed proposed work.

2.1 Outdoor Testbed

The outdoor testbed is deployed in a field near the campus of the University of Illinois at Urbana-Champaign. This testbed is motivated by high-end environmental monitoring applications, such as recording bird vocalizations in CD-quality and bird nesting activities by using infrared cameras. The data collected on the test have been used by our collaborators in the department of natural resources and environmental sciences for bird habitat studies.

Currently, 9 nodes have been deployed. Figure 2.1 displays the map of the current deployment of the outdoor testbed. Figure 2.2 provides an outside look at one node. Every component, except the solar panels and the antenna, is packed in a waterproof enclosure. The components inside of the enclosure are shown in Figure 2.3. In Figure 2.4, we sketch the basic components of each node, which can be grouped into two subsystems: energy subsystem and computing subsystem. We next describe them in detail respectively.
2.1.1 Energy Subsystem

Each node is powered by one 12 Volt deep cycle battery (DEKA 8G31) with a capacity of 98 AH. Although they look like normal car batteries, deep cycle batteries are especially designed for prolonged discharges at lower current, while car batteries are designed for high initial cranking current for a short time. A set of two solar panels [39] are used on each node to charge the battery, and the output of each panel is rated as 120 Watts. Note that, this wattage specification of solar panels is rated under an ideal environment, where the sun is directly overhead at local noon and the atmosphere is clear and dry. The real output of the panels depends on the season, the weather, the mounting angle, and the latitude where they are going to be deployed. The location of our deployment is at (40.10N, 88.20W). After a
period of calibration, we decided to use two panels to support each node. Both panels are set facing south and have a 40° angle with the ground. So they are approximately perpendicular to the sunlight during both spring and fall, which yield higher yearly energy collection.

In order to avoid overcharging the battery and causing potential electrical damage to the devices, a charge controller (Xantrex C35 [40] in charge control mode) is used to regulate the output voltage and current of the panels down to what the battery needs at the time. The whole computing subsystem draws power from the battery, with a load controller (Xantrex C35 in load control mode) sitting in between, which disconnects the circuit when the battery is nearly empty to protect the battery from over discharging.

2.1.2 Computing Subsystem

Since the testbed aims at supporting a wide spectrum of applications, including very high-performance sensing and processing (e.g., record CD-quality acoustic data at a rate of 705.6 Kbps), we investigated several PCs, looking for powerful computing capabilities and reasonably large local storage as well as high power efficiency. We eventually chose Asus EEE
Figure 2.4: Basic components of each node.

PCs [41]. An EEE PC is equipped with an Intel 900 MHz CPU, 1GB DDR2, 20GB solid static disk and 3 USB ports. It can run Linux or Windows. The PC consumes only about 10 Watts under moderate load and 15 Watts when heavily loaded (namely, 0.8 Ampere and 1.2 Ampere respectively drained from the 12 Volt battery).

An EEE PC has a built-in wireless interface, which draws about 2.4 Watts but has a very limited transmission range, especially when sealed inside the waterproof enclosure. Hence, we decided to use a stand-alone wireless device to provide communication between nodes. The Linksys WRT54GL router [42] was selected because of its low power consumption (6 Watts) and support for third-party firmware, which allows us to program the device. Moreover, its antenna is detachable. By extending the antenna outside of the waterproof enclosure, we achieved at least a 3 Mbps transmission rate at a distance of 50 meters outdoors. For each Linksys WRT54GL router, we update its firmware as openwrt [43], a Linux distribution for embedded devices. Openwrt not only provides more device customization choices, but also supports running programs compiled for the MIPS architecture of WRT54GL. By using openwrt, we configure the wireless interface of each router to work in an ad-hoc mode with a
static and unique virtual IP. We also assign a static virtual IP (but in a different subnet from the ad-hoc wireless network) to the Ethernet interface of each EEE PC, and connect it to the router’s Ethernet interface. On each EEE PC, a permanent route is set for packets going to the ad-hoc subnet use the router’s Ethernet interface as the next hop. The router then checks its routing table and forwards the packets accordingly for the EEE PC. A routing daemon runs on each router to monitor active neighboring routers and update its routing table. Besides, port forwarding is setup on each router so that packets can be forwarded from the ad-hoc subnet to EEE PCs.

Additional hardware is also employed to provide information required for energy management, such as the remaining energy in the battery and the charging rate from solar panels. The remaining energy of a battery can be approximately indicated by its voltage. A Phidget precision voltage sensor [44] is wired in parallel with the battery to measure its voltage. In the meantime, the charging current from the solar panels is measured by a Phidget current sensor [45] in series between the Charge Controller and the battery. Both the voltage sensor and the current sensor are connected to a Phidget InterfaceKit [46], which converts analog readings of the sensors to digital and then feed them to the EEE PC through a USB port.

Like other wireless devices, WRT54GL routers in idle mode consume non-trivial power. Thus, SolarStore is allowed to completely turn the router off when it is necessary to save energy. We note that turning on or off a WRT54GL router can only be done through switching on or off its power supply. Thus, we use a dual-coil latching relay, controlled through the Phidget InterfaceKit, to connect or disconnect the power supply of the router. The advantage of dual-coil latching relays over normal relays is that they need only a short current pulse instead of a continuous current in order to keep the circuit open or close, which leads to further energy savings.

It is not straightforward to reboot an EEE PC back on after it shuts down when the battery is depleted. An auto reboot timer on its motherboard can be used to wake up an EEE PC. Therefore, we have a daemon running on each EEE PC to refresh the auto reboot
timer every 10 minutes, while the timer is set to 20 minutes. Hence, normally, the timer will be reset before it goes off. If an EEE PC is down because the battery is depleted, the timer will continue counting down, freeze at zero, and immediately go off when the power is back on. This way, the whole system can work quite autonomously without need of maintenance.

2.2 Indoor Testbed

Experimental results on the testbed greatly depend on the outdoor environment, especially weather conditions during the experiments. Because we target on solar-powered systems with a long time of operation, it is difficult, if not impossible, to have repeated weather conditions so as to compare our proposed schemes with other schemes. Moreover, given the fixed location of our testbed, some environmental conditions may never be attained locally. Therefore, an indoor solar system testbed has been set up to conduct a fair performance evaluation under a wide range of environmental conditions.

Figure 2.5 is a snapshot of the 9 nodes in our indoor testbed. The computing subsystem of each node is a clone of the one in the outdoor testbed, including an EEE PC, a Linksys WRT54GL wireless router and a PC-controlled power switch.

As for the energy subsystem, a solar panel emulator is used to emulate the charging current
from solar panels based on real solar energy traces collected on the outdoor testbed. Taking the charging current as one of its inputs, and leveraging Phidget current sensors to obtain the power consumption rate of the computing subsystem, a battery emulator maintains the residual energy level of the battery for each node. Even though nodes are now powered by normal indoor AC supplies, the battery emulator will disconnect the AC power for a node if the residual energy in its emulated battery reaches zero, and connect the power back on when the residual energy becomes positive again.

The automatic control of the AC power supply for each node is realized through X10 modules, which are typically used to control appliances in smart houses. The AC/DC power adapter of each node connects to an X10 module first before being plugged into an AC outlet. Each X10 module listens for incoming X10 commands from the power line, and then switches the circuit on/off if X10 on/off commands addressed to this module are received. In addition, an X10 controller is used by the battery emulator to issue X10 commands onto the power line.

With regard to communication, indeed nodes in the indoor testbed are so close to each other that every node is just one hop away. We leverage MAC address filters on the WRT54GL routers to form exactly the same network topology as measured in the outdoor testbed (Figure 2.1). Moreover, it is known that indoor wireless links behave very differently from outdoor links in terms of such as signal to noise ratio and packet loss distribution. However, the data exchange between nodes is implemented by using TCP, and thus we mainly focus on emulating the TCP bandwidth of the outdoor links in the indoor testbed. Honestly, it is hard to achieve identical between the indoor and outdoor testbeds. But by adjusting the transmit power of the WRT54GL routers, the TCP bandwidth of the indoor and outdoor links is in a range of 3.0 Mbps to 4.5 Mbps.
Chapter 3

Utilizing Erasure Codes for Reliable Data Delivery

3.1 Introduction

Because of the error-prone nature of wireless communication, a spectrum of solutions has been proposed at every layer of the network protocol stack on how to reliably deliver sensory data in wireless sensor networks. Among them, a class of approaches is to proactively add redundancy by using simple duplication or advanced coding schemes (e.g. erasure coding [47, 48, 49, 50]), and send multiple copies of a message simultaneously to mitigate the effects of single-message losses. Due to the limited energy in traditional battery-powered sensor networks, exploiting any level of redundancy will inevitably reduce the network lifetime. This is because achieving redundancy takes extra energy and the total amount of work that can be accomplished by a node is approximately pre-determined by the initial energy in its battery. Therefore, most prior efforts mainly focus on the trade-off between reliability and network lifetime.

However, this dilemma could be relieved in wireless sensor networks with renewable energy sources. Considering the fact that a full battery can not harvest more energy, there is an incentive to spend energy to make room to harvest more energy. As a result, the extra spending has no impact on the node lifetime since this energy surplus would be wasted otherwise. In this chapter, we consider the problem of utilizing such energy surplus (if any) in solar-powered sensor networks to adaptively adjust the redundancy level of erasure codes used in communication, so that the data delivery reliability is improved while the network lifetime is still conserved.
As an efficient technique for recovering data from partial loss or corruption, erasure coding [47, 48, 49, 50] has long been adopted for peer-to-peer systems [51] and delay tolerant networks [52, 53], coping with the failure of packet transmissions [47, 49] or the breakdown of storage systems [54, 55]. An erasure coding scheme partitions a message into blocks and then transforms them into a large number of encoded blocks. The original message can be recovered as long as enough encoded blocks are received. The more encoded blocks are generated and transmitted, the more likely it is that the original message can be recovered. In the context of sensor networks, the redundancy level (or called replication factor) should be adjusted dynamically according to the energy availability of sensor nodes.  

The uniqueness and challenge of this problem can be easily illustrated by a simple example. Consider the topology shown in Figure 3.1, which also shows the changes in the residual energy of node \( n_3 \) in one day. As we can see, \( n_3 \) is fully charged around noon and stays in the fully charged state until 3PM because the energy charging rate is higher than the consumption rate during these 3 hours. This makes it possible for node \( n_3 \) to spend energy at a higher rate during this period (i.e., increase its communication redundancy) and still remain in the fully charged state at 3PM, as long as the total amount of the extra spending is less than the energy surplus. As a result, the data delivery reliability is improved and the network lifetime remains intact.

\[ ^1 \text{Retransmission is another option for reliable communication, and we will give a discussion about it in Section 3.2.} \]
A naive approach for utilizing energy surplus is to add communication redundancy only when the battery is full. However, under this simple approach, erasure coding is only active during the short periods of the fully charged state. Consequently, the energy surplus may not be fully utilized because adding more redundancy will have a very marginal gain if the redundancy level is already reasonably high for these periods. Therefore, it could be better for a node to start increasing the redundancy level even before reaching the fully charged state.

Furthermore, from the perspective of end-to-end flows, a node (e.g., $n_1$ or $n_2$ in this example) should not arbitrarily increase its transmission redundancy even though it has plenty of energy surplus. This is because it will take the receiving node (e.g., $n_3$) extra energy to receive the redundant communication. In this example, if link $(n_3, n_4)$ has a low quality, it may be better for $n_3$ to allocate most of its energy surplus for transmitting on link $(n_3, n_4)$ than receiving on link $(n_1, n_3)$. As a relay node, another problem that it has to face is how to divide its energy surplus for the passing flow such that the performance of the whole network is improved.

This example shows that even for simple topologies determining the optimal redundancy levels can be complex. The optimal levels depend not only on the network properties (e.g., topology, flow routes and link qualities), but also on the solar energy harvesting process. In this chapter, we rigorously formulate an optimization problem to determine how to dynamically adjust the redundancy level of each data link over the time period of interest, such that (1) the end-to-end packet delivery probability is maximized and (2) the network lifetime is not affected.

The formulated optimization problem, however, is in general hard to solve because of the combinatorics involved and the special curvature of its objective function. By exploiting special properties of the problem, we propose an effective approximated solution called SolarCode. We prove that SolarCode has a constant approximation ratio. Moreover, we also prove that the combinatoric functions involved in the objective are guaranteed to be
concave, as long as the quality of considered links is not too low (e.g., above 30% delivery rate). Therefore, SolarCode solves the problem by using general convex optimization techniques.

The remainder of this chapter is organized as follows. In Section 3.2, we introduce the system model, formulate the problem and discuss the difficulties in finding its optimal solutions. In Section 3.3, we propose our approximated solution, SolarCode. Then, we address implementation issues in Section 3.4, evaluate SolarCode in Section 3.5, and finally conclude the chapter by Section 3.6.

3.2 System Model and Problem Formulation

In this section, we rigorously formulate an optimization problem that determines the redundancy level of each link in a solar-powered sensor network such that the total end-to-end packet delivery probability is maximized while the network lifetime is conserved.

3.2.1 Network Model

We consider a network with a node set $\mathcal{N}$ and a link set $\mathcal{L}$. Let $l(i, j) \in \mathcal{L}$ be a directional link from node $i$ to node $j$ where $i, j \in \mathcal{N}$. Sensory data are generated on nodes in a subset $\mathcal{S} \subseteq \mathcal{N}$, and then forwarded through the network to particular destinations. The traffic pattern that we consider in this work is general so that different data flows could have a single or multiple destinations, depending on the applications. Solely for the sake of presentation, we assume single path communication, and that the route used by a flow is static throughout the time period under consideration. Moreover, we assume the route of a flow is determined by some other routing module and is not considered as an optimization knob in this work. Let $r_s$ be the rate of the sensory data generated on node $s \in \mathcal{S}$, and $f_s$ be the route used to forward these data in the network. We write $l(i, j) \in f_s$ to denote that link $l(i, j)$ is on the route $f_s$. 
Communication on wireless links is error-prone. Errors may result from a variety of effects, such as interference, fading and background noise coming from many sources (e.g., solar radiation). Many wireless link error models have been proposed to characterize these effects. We adopt an effective and widely used statistical BER-based (Bit Error Rate) model. The transmission of a packet is successful only when all its bits are received correctly. Thus, the successful transmission probability for a packet of $m$ bits is $(1 - p_e)^m$, where $p_e$ is the statistical bit error rate on this link.

### 3.2.2 Erasure Coding Model

When erasure coding is employed, a packet is first divided into $b$ blocks and then encoded into $\alpha b$ code blocks such that if $b$ or more code blocks are received, the original packet can be decoded. The parameter $\alpha$ determines the degree of redundancy and is called the replication factor. Choices of $b$ could also affect the coding and decoding process and hence the overall data delivery performance. However, for the sake of simplicity, we have chosen to fix this parameter in our formulation, and then try to determine the replication factor $\alpha$ in order to maximize the end-to-end packet delivery probability.

Denote $p = (1 - p_e)^{\frac{m}{b}}$ as the successful transmission probability of each code block. With a replication factor $\alpha$, $\alpha b$ code blocks are transmitted over a link and we can express the successful decode probability as a function of $\alpha$:

$$\tilde{Pr}(\alpha) = \sum_{k=\lfloor \alpha b \rfloor}^{\alpha b} \binom{\alpha b}{k} p^k (1 - p)^{\alpha b - k}. \quad (3.1)$$

Note that $\lfloor \alpha b \rfloor$ in Eq (3.1) has to be an integer. We can allow $\alpha$ to be any real number in $[1, +\infty)$, by using a coding module to always generate $\lfloor \alpha b \rfloor$ code blocks, and generate one extra code block with probability $\alpha b - \lfloor \alpha b \rfloor$. Thus, the successful decoding probability function for a general $\alpha$ is

$$Pr(\alpha) = (1 - \alpha b + \lfloor \alpha b \rfloor) \tilde{Pr}(\lfloor \alpha b \rfloor) + (\alpha b - \lfloor \alpha b \rfloor) \tilde{Pr}(\lfloor \alpha b \rfloor). \quad (3.2)$$
When $\alpha = 1$, it means that no redundancy is introduced and thus we have $Pr(\alpha = 1) = (1 - p_e)^m$. One can prove that $Pr(\alpha)$ is a non-decreasing function of $\alpha$. This is intuitively true as the more redundancy that is used, the higher successful probability that can be achieved. However, achieving redundancy takes extra energy and thus the replication factor $\alpha$ should be adapted according to time-varying energy constraints.

### 3.2.3 Energy Model

Each node $i \in \mathcal{N}$ is powered on a rechargeable battery with capacity $B_i$. And the battery is charged by a solar panel. The solar energy available for harvesting depends on many factors. Physical models have been proposed to quantify the solar energy for given times, dates, geographical locations as well as weather conditions. In section 3.4, we will elaborate on how we project the available solar energy based on historical solar energy traces and weather forecast information. Now in our formulation, we assume that the solar energy at time $t$ for node $i$ is known as an input, denoted as $S_i(t)$.

Let $C_i$ be the CPU power consumed by applications together with system processes (e.g., a routing daemon) running on node $i$. The wireless radio of node $i$ has a power consumption rate $P_{TX}^i$ for transmitting and $P_{RX}^i$ for receiving. Hence, the power consumption rate for transmitting and receiving data through link $l(i, j)$ when applying erasure coding is $\alpha_{ij}(t) P_{TX}^i$ and $\alpha_{ij}(t) P_{RX}^j$, respectively, where $\alpha_{ij}(t)$ is the replication factor used for link $l(i, j)$ at time $t$.

Note that $\alpha_{ij}(t)$ is an absolute factor with respect to the raw data packet. Therefore, the overall power consumption rate of node $i$ at time $t$ is:

$$W_i(t) = C_i + \sum_{l(i,j)} R_{ji}(t)\alpha_{ji}(t) P_{RX}^i + \sum_{l(i,j)} R_{ij}(t)\alpha_{ij}(t) P_{TX}^i,$$  \hspace{1cm} (3.3)

where $R_{ij}(t) = \sum_{s \in \mathcal{I}(i,j) \in f_s} r_s$ is the total traffic rate of raw data on link $l(i, j)$ at time $t$. 

3.2.4 Problem Formulation

For the ease of formulation, we discretize the time under consideration, $T$, into $N_t$ slots. Each of duration is $\Delta = T/N_t$. Our objective is to adapt the replication factor $\alpha_{ij}(t)$ of each link $l(i,j)$ such that the end-to-end packet delivery probability weighted by flow rates throughout $T$ is maximized.

When a packet is relayed by a node $i$ to a node $j$, it is encoded into $\alpha_{ij}(t)b$ blocks and all blocks are sent on link $(i,j)$. The packet can be successfully decoded on node $j$ with probability $Pr(\alpha_{ij}(t))$. If successfully decoded, this packet will then be encoded again into $\alpha_{jk}(t)b$ blocks and sent on link $(j,k)$ to its next hop $k$. Then the end-to-end packet delivery probability for a flow $f_s$ is $D_s(t) = \prod_{l(i,j) \in f_s} Pr(\alpha_{ij}(t))$.

Let $R_S = T \sum_{s \in S} r_s$ be the total traffic load. Therefore our objective function is

$$\max \sum_{s \in S} \sum_{t=1}^{N_t} \frac{r_s \Delta}{R_S} \prod_{l(i,j) \in f_s} Pr(\alpha_{ij}(t)),$$

subject to the following constraints.

First, our goal is to utilize the energy surplus from a renewable energy source to enhance the data delivery reliability. It has to be ensured that the extra energy spending does not affect the network lifetime. Thus, the residual energy of node $i$, denoted as $e_i(t)$, should satisfy a non-blackout constraint:

$$e_i(t) > 0, \quad \forall i \in \mathcal{N}, 1 \leq t \leq N_t. \quad (3.5)$$

In order to have feasible solutions, we assume that this non-blackout constraint is satisfied when all $\alpha_{ij}(t)$s equal to one. It means that the system does not have blackout originally when erasure coding is not used. This can be enforced in the design of the actual system.

Second, the residual energy of node $i$ at time slot $t$ (i.e., $e_i(t)$) equals to the remaining energy at the last time slot $(t-1)$ plus the solar energy harvested subtracting the consumed
energy $W_i(t)\Delta$. Note that the available solar energy $S_i(t)$ may not be fully harvested into the battery because of the battery capacity bound. Namely, $e_i(t)$ should also be bounded by the battery capacity $B_i$. Thus, we have an energy evolution constraint:

$$e_i(t) = \min\{e_i(t - 1) + (S_i(t) - W_i(t))\Delta, B_i\}, \forall i \in \mathcal{N}, 1 \leq t \leq N_i. \quad (3.6)$$

Observe that the objective function is non-decreasing with $e_i(t)$, which means that the more energy is in the battery, the more is the energy surplus, and the higher is the reliability that can be achieved. Therefore, the energy evolution constraint is equivalent to the following two linear constraints:

$$e_i(t) \leq e_i(t - 1) + (S_i(t) - W_i(t))\Delta, \quad (3.7)$$

$$e_i(t) \leq B_i, \quad \forall i \in \mathcal{N}, 1 \leq t \leq N_i. \quad (3.8)$$

We may also desire the residual energy of each node at the end of $T$ to be above some threshold so that the system would have reasonable initial energy for the next experiment. This can be realized by using a residual energy constraint:

$$e_i(N_t) \geq R E_i, \quad \forall i \in \mathcal{N}, \quad (3.9)$$

where $R E_i$ is the residual energy threshold of node $i$.

In addition to the above constraints, replication factors $\alpha_{ij}(t)$ have to be greater than or equal to one in order to produce meaningful erasure codes:

$$\alpha_{ij}(t) \geq 1, \quad \forall (i, j) \in \mathcal{L}, 1 \leq t \leq N_i. \quad (3.10)$$

Finally, the initial energy $e_i(0) \ (i \in \mathcal{N})$ are given as inputs. Recall that the available solar energy $S_i(t) \ (i \in \mathcal{N}, 1 \leq t \leq N_i)$ are also assumed to be known as inputs of the problem.
3.2.5 Discussion

Comparing to adding redundancy, retransmission is another option for reliable data communication, but usually has a longer delay because of a relatively long time-out duration involved. Therefore, we concentrate on using erasure codes for reliable data delivery in this work. In fact, our approach can be extended to retransmission schemes, where $\alpha$ can be regarded as an upper bound for the number of retransmissions.

Aside from the extra energy cost, a higher replication factor also causes higher interference overhead and eventually could dramatically bring down the link quality. One can formulate a network cross-layer optimization problem to take into account the effect of replication factors on communication interference. However this would lead to a more complex problem that may have no effective solution. Hence, signal interference can be considered to have nearly no impact on link quality, by assuming that the redundant traffic load is much lower than the link capacity. This is usually true for sensor networks, where raw traffic load is low in most applications.

3.3 SolarCode: an Effective Approximated Solution

The formulated optimization problem is in general hard to solve due to the complexity in its objective function. First, the link delivery probability $Pr(\alpha_{ij}(t))$ (Eq (3.2)) in the objective function is a summation of a series of combinatorial terms and the number of the terms also depends on the parameter $\alpha_{ij}(t)$. Even though $Pr(\alpha_{ij}(t))$ is a non-decreasing function of $\alpha_{ij}(t)$, its curvature is neither concave nor convex, and thus we can not solve the problem by using convex optimization techniques. Moreover, although a concave function may be used to approximate $Pr(\alpha_{ij}(t))$, the end-to-end packet delivery probability of a route relies on the product of $Pr(\alpha_{ij}(t))$ of each link on the route. No general method has been proposed for optimizing products of concave functions, because no conclusion can be drawn on the curvature of products of concave functions.
Therefore, in this section, we propose an effective approximated solution called SolarCode, and prove that it has a constant approximation ratio. SolarCode transforms the formulated problem into a convex optimization problem, which can be solved by using general optimization techniques in either a centralized or a decentralized manner.

### 3.3.1 A Close Lower Bound of the Objective

As mentioned above, one difficulty in solving the optimization problem is from the products of probability functions $Pr(\alpha_{ij}(t))$ in Eq (3.4). A natural way to bypass this is to use a logarithm function to transform the products of $Pr(\alpha_{ij}(t))$ into summations of $\log(Pr(\alpha_{ij}(t)))$.

Maximizing Eq (3.4) is equivalent to maximizing its logarithm value. Notice that

$$\sum_{s \in S} \sum_{t=1}^{N_t} \frac{r_s \Delta}{R_s} = 1.$$ 

Thus we can obtain an lower bound on the objective function by applying the concave property of logarithm functions,

$$\log \left( \sum_{s \in S} \sum_{t=1}^{N_t} \frac{r_s \Delta}{R_s} \prod_{l(i,j) \in f_s} Pr(\alpha_{ij}(t)) \right) \geq \sum_{s \in S} \sum_{t=1}^{N_t} \frac{r_s \Delta}{R_s} \cdot \log \left( \prod_{l(i,j) \in f_s} Pr(\alpha_{ij}(t)) \right)$$

(3.10)

$$= \sum_{s \in S} \sum_{t=1}^{N_t} \frac{r_s \Delta}{R_s} \cdot \sum_{l(i,j) \in f_s} \log \left( Pr(\alpha_{ij}(t)) \right).$$

(3.11)

Recall $D_s(t) = \prod_{l(i,j) \in f_s} Pr(\alpha_{ij}(t))$. The equality in (3.10) holds when all $D_s(t)$s are equal for $\forall s \in S$ and $1 \leq t \leq N_t$. But there is no general upper bound for the difference between the two sides of this inequality. In particular, it becomes infinity when any $D_s(t)$ becomes infinitesimal. Considering that $D_s(t)$ is the end-to-end delivery probability of a flow, we have $\beta = \inf\{D_s(t) : s \in S, 1 \leq t \leq N_t\} > 0$, as long as the flow paths picked by the routing
algorithm are not broken. Thus, we can obtain a finite upper bound for the logarithm approximation above.

**Theorem 1.** If \(0 < \beta \leq D_s(t) \leq 1\) (for all \(s \in \mathcal{S}\) and \(1 \leq t \leq N_t\), then

\[
0 \leq \log\left(\sum_{s \in \mathcal{S}} \sum_{t=1}^{N_t} \frac{r \Delta}{R_s} D_s(t)\right) - \sum_{s \in \mathcal{S}} \sum_{t=1}^{N_t} \frac{r \Delta}{R_s} \log(D_s(t)) \leq \log(\Lambda),
\]

where \(\Lambda = (\beta - 1)/((\beta^{1/\ln \beta + 1/(1-\beta)}) \ln \beta)\).

**Proof.** The lower bound holds because \(\log(\cdot)\) is concave. To prove the upper bound, it is equivalent to find the maximum value of

\[
\log\left(\frac{\sum_{s \in \mathcal{S}} \sum_{t=1}^{N_t} \frac{r \Delta}{R_s} D_s(t)}{\prod_{s \in \mathcal{S}} \prod_{t=1}^{N_t} D_s(t) \frac{r \Delta}{R_s}}\right) := \log(g).
\]

Due to the symmetry of \(D_s(t) (1 \leq t \leq N_t)\) in function \(g\), \(D_s(t)\) will be the same for all \(t\) when \(g\) is maximized. Letting \(D_s(t) = x_s\), we have \(g = \log((\sum_{s \in \mathcal{S}} N_t x_s)/\prod_{s \in \mathcal{S}} x_s)\).

Note that the variables of function \(g\) are independent, and thus we can seek its optimal value by tuning the variables one by one. Therefore, for each \(x_i (i \in \mathcal{S})\), we have

\[
\frac{\partial g}{\partial x_i} = \frac{N_t \frac{r \Delta}{R_s} x_i^{N_t \frac{r \Delta}{R_s} - 1} (x_i - \sum_{s \in \mathcal{S}} N_t \frac{r \Delta}{R_s} x_s)}{x_i^{N_t \frac{r \Delta}{R_s}} \prod_{s \in \mathcal{S}} N_t \frac{r \Delta}{R_s} x_s}.
\]

If \(x_i \geq \sum_{s \in \mathcal{S}} N_t \frac{r \Delta}{R_s} x_s\), the above partial derivative is non-negative. Moreover, the condition \(x_i \geq \sum_{s \in \mathcal{S}} N_t \frac{r \Delta}{R_s} x_s\) keeps true as \(x_i\) increases. Therefore, \(g\) is a non-decreasing function of \(x_i\), which means \(g\) is maximized at the maximum value (i.e., 1) of \(x_i\).

On the other hand, if \(x_i \leq \sum_{s \in \mathcal{S}} N_t \frac{r \Delta}{R_s} x_s\), \(g\) is a non-increasing function of \(x_i\), which means \(g\) is maximized at the minimum value (i.e., \(\beta\)) of \(x_i\). Therefore, \(g\) is maximized when some \(x_i (i \in \mathcal{A} \subseteq \mathcal{S})\) are \(\beta\), while other \(x_i (i \in \mathcal{S} - \mathcal{A})\) are 1. Denoting \(y = \sum_{s \in \mathcal{A}} N_t \frac{r \Delta}{R_s}\),
we have $\sum_{s \in S-A} N_t \frac{r_s \Delta}{K_s} = 1 - y$ because $\sum_{s \in S} \sum_{t=1}^{N_t} \frac{r_s \Delta}{K_s} = 1$, and then we need to further optimize

$$g(\{ x_s = \beta : s \in A; \} , \{ r_s \}) = \frac{\beta y + (1 - y)}{\beta y} := h(y),$$

with respective to $y$. Solving $\dot{h}(y) = 0$, we obtain the optimal $y^* = 1/\ln \beta + 1/(1 - \beta)$. Plugging $y^*$ into $h(y)$, we have the maximum of $h$ as $(\beta - 1)/[(\beta^{1/\ln \beta+1/(1-\beta)}) \ln \beta]$, which is also the maximum of $g$. 

Theorem 1 implies that the approximation error introduced by the logarithm transform can be bounded by a constant $\log(\Lambda)$. Converting its logarithm value back to the original objective, we have that the ratio between the optimal and approximated values is bounded by $\Lambda$. More importantly, this approximation ratio is independent of the problem size, including the network size, flow rate the time slot granularity.

### 3.3.2 Concavity of $Pr(\alpha)$

Now we deal with the difficulty brought by the curvature of the probability distribution function $Pr(\alpha)$. For the easy of presentation, we omit $\alpha$’s subscripts of links and times in this subsection.
It can be easily verified that $Pr(\alpha)$ ($\alpha \geq 1$) is not always concave or convex, as shown in Figure 3.2. Note that $Pr(\alpha)$ also depends on the number of code blocks $b$ that each packet is divided into, and the block delivery probability $p$. Interestingly, one insight that can be observed from Figure 3.2 is that $Pr(\alpha)$ tends to become concave as $p$ goes up. Thereby, we hypothesize that $Pr(\alpha)$ is concave when the quality of links used by flows is reasonably good. This is confirmed by the following theorem.

**Theorem 2.** $Pr(\alpha)$ is always concave with respect to $\alpha$ if the packet delivery probability

$$p_m = p^b \geq \left[ \frac{b^2 - 1 + \sqrt{3(b^2 - 1)}}{(b + 1)(b + 2)} \right]^b \tag{3.12}$$

*Proof.* As in Eq (3.2), $Pr(\alpha)$ is on the line segment connecting point $([\alpha b]/b, \widetilde{Pr}([\alpha b]/b))$ and $([\alpha b]/b, \widetilde{Pr}([\alpha b]/b))$, when $\alpha b$ is not an integer. Then it is sufficient to prove that the slope of the line segment between consecutive points $\alpha = i/b$ ($i = b, b + 1, ...$) is always decreasing under condition (3.12).

Let $\alpha = i/b$, then the slope of the linear segment between $\widetilde{Pr}((i - 1)/b)$ and $\widetilde{Pr}(i/b)$ is $b[\widetilde{Pr}(i/b) - \widetilde{Pr}((i - 1)/b)]$. Thus we want

$$\widetilde{Pr}(\frac{i}{b}) - \widetilde{Pr}(\frac{i - 1}{b}) \geq \widetilde{Pr}(\frac{i + 1}{b}) - \widetilde{Pr}(\frac{i}{b}).$$

Substituting $\widetilde{Pr}(\alpha) = \sum_{k=b}^{\alpha b} \binom{\alpha b}{k} p^k (1 - p)^{\alpha b - k} = 1 - \sum_{k=0}^{b-1} \binom{\alpha b}{k} p^k (1 - p)^{\alpha b - k}$ in, we obtain

$$\sum_{k=0}^{b-1} Q_k(i) \binom{i}{k} p^k (1 - p)^{i - k} \geq 0,$$

where $Q_k(i) = \frac{(i+1)(1-p)}{i+1-i-k} + \frac{i-k}{i(1-p)} - 2$. It is then sufficient to show each $Q_k(i) \geq 0$ under condition (3.12), for $0 \leq k \leq b - 1$ and $i = b + 1, b + 2, ...$. By solving the inequality

27
\( Q_k(i) \geq 0 \) with respect to \( p \), we have

\[
p \geq \frac{ki + \sqrt{ki(i + 1 - k)}}{i(i + 1)}
\]

It is easy to verify that the right hand of the inequality above increases with \( k \) and decreases with \( i \). Plugging in the largest \( k = b - 1 \) and the smallest \( i = b + 1 \), we obtain the condition (3.12).

Note that \( p_m = p^b \) is the successful delivery probability for a packet, having a size equal to \( b \) encoded blocks, when no coding scheme is used. Although the lower bound in Theorem 2 is not tight, it is always below 29\% even for very large \( b \), as shown in Figure 3.3. Considering that \( b \) is usually not too large because of the extra framing overhead introduced by encoded blocks, we can see that this lower bound on the link quality can be easily satisfied by asking the routing scheme to choose links with reasonable packet delivery probability (e.g., above 28\% for \( b = 8 \)).

3.3.3 Centralized Solution

Based on these two theorems, we can transform the original problem into a convex optimization problem, which can be solved by using convex optimization solvers in a centralized manner.

Algorithm 1 outlines our centralized version of SolarCode. It first retrieves weather forecast information from online weather services, and uses a pre-trained model (will be elaborated in Section 3.4) to predict the solar energy input. Then, it collects the quality \( p \) of each link, based on which it computes the successful decoding probability \( \tilde{Pr}(\cdot) \) for a large number \( Z \) of integer points (usually \( Z = 3b \) is large enough). Assuming condition (3.12) in Theorem 2 is satisfied, we can then express the general \( Pr(\alpha) \) as

\[
Pr(\alpha) = \inf \{ \ell_i(\alpha), 1 : i = b, ..., Z - 1 \},
\]

(3.13)
where \( \ell_i(\cdot) \) is the line function determined by the two points \((i/b, \tilde{Pr}(i/b))\) and \(((i+1)/b, \tilde{Pr}((i+1)/b))\), which can be expressed as:

\[
\ell_i(\alpha) = \frac{b}{\tilde{Pr}(i/b) - \tilde{Pr}((i+1)/b)} \cdot \alpha + \frac{(i+1)b}{\tilde{Pr}((i+1)/b)} - \frac{ib}{\tilde{Pr}(i/b)}.
\]

Plugging Eq (3.13) into the logarithm objective function Eq (3.11), we obtain a normal convex optimization problem, and solve it for \( \alpha_l(t) \), \( l \in \mathcal{L} \) and \( 1 \leq t \leq N_t \). Finally, the solution \( \alpha_l(t) \) is passed onto each node, which uses it to adjust redundancy in the communication.

Suppose that transmitting one real number requires one message unit, the communication overhead of centralized SolarCode consists of \(|\mathcal{L}| + |\mathcal{N}|\) message units used to collect the link quality of each link and the initial battery state of each node \(^2\) and \(N_t|\mathcal{L}|\) messages units used to disseminate the solution. Let \( H \) be the average number of hops between the basestation and every node. The overall message overhead is thus \([(N_t + 1)|\mathcal{L}| + |\mathcal{N}|]H\), which increases with the size the network. Therefore, we propose a decentralized SolarCode that only requires message exchanges between neighboring nodes.

### 3.3.4 Decentralized Solution

We decentralize Algorithm 1 as follows. First, each node computes Eq (3.13) for each of its outgoing link. Second, for solving the optimization problem, the gain of erasure codes on each link in the objective function can be decoupled after the logarithm transform. Now the only coupling between nodes comes from the energy consumption in data communication. As in constraint (3.7), data communication costs energy of both the sender and receiver, and thus the replication factor should be adjusted according to the energy status of both the sender and receiver. To decouple this constraint, we introduce lagrangian multipliers \( \mu^l_i \)

\(^2\)we assume that the battery capacity of each node is known to SolarCode and thus does not need to be collected.
Algorithm 1 SolarCode ($\mathcal{L}, \{S_i(t) : i \in \mathcal{N}, 1 \leq t \leq N_t\}$)

1: Retrieve weather forecast and predict solar energy input $\{S_i(t) : i \in \mathcal{N}, 1 \leq t \leq N_t\}$;
2: for $i \in \mathcal{N}$ do
3:   Collect initial residual energy $e_i(0)$;
4: end for
5: for $l \in \mathcal{L}$ do
6:   Collect link quality information $q$;
7:   Compute $Pr(l_i)$ for $i = b, ..., Z$;
8: end for
9: Solve the optimization problem with objective Eq (3.11), where $Pr(\alpha_l(t)) \equiv \inf\{\ell_i(\alpha_l(t)), 1 : i = b, ..., Z - 1\}$, subject to constraint (3.5)(3.7)(3.8)(3.9) with input $\{S_i(t)\}$ and $\{e_i(0)\}$;
10: for $i \in \mathcal{N}$ do
11:   Pass $\alpha_l(t) : l(i, j) \in \mathcal{L}, 1 \leq t \leq N_t$ to node $i$;
12: end for

for constraint (3.7), and we have the Lagrangian of original optimization problem\(^3\)

$$L(\alpha, e, \mu) = \sum_{s \in \mathcal{S}} \sum_{t=1}^{N_t} r_s \cdot \sum_{l(i,j) \in f_s} \log(P_r(\alpha_{ij}(t))) - \sum_{i \in \mathcal{N}} \sum_{t=1}^{N_t} \mu_t \left[ e_i(t) - e_i(t - 1) - \left( S_i(t) - C_i - \sum_{l(j,i)} R_{ji}(t) \alpha_{ji}(t) P_{RX}^i - \sum_{l(i,j)} R_{ij}(t) \alpha_{ij}(t) P_{TX}^i \right) \Delta \right].$$

Since it can be derived that\(^4\)

$$\sum_{t=1}^{N_t} \mu_t^t \left[ e_i(t) - e_i(t - 1) \right] = \sum_{t=1}^{N_t} e_i(t) (\mu_t^t - \mu_t^{t+1}) - \mu_t^1 e_i(0)$$

and

$$\sum_{i \in \mathcal{N}} \sum_{t=1}^{N_t} \mu_t^t \left( \sum_{l(j,i)} R_{ji}(t) \alpha_{ji}(t) P_{RX}^i + \sum_{l(i,j)} R_{ij}(t) \alpha_{ij}(t) P_{TX}^i \right) \Delta$$

$$= \sum_{i \in \mathcal{N}} \sum_{t=1}^{N_t} \sum_{l(i,j)} R_{ij}(t) \alpha_{ij}(t) (\mu_j^t P_{RX}^j + \mu_i^t P_{TX}^i) \Delta,$$

\(^3\)We omit the constant $\Delta/R_S$ in Eq (3.4) as it has no effect on the optimization.

\(^4\)We define $\mu_i^{N_t+1} = 0$ for the ease of presentation.
we can reorganize the Lagrangian as follows:

\[
L(\alpha, e, \mu) = \sum_{s \in S} \sum_{t=1}^{N_t} r_s \cdot \sum_{l(i,j) \in f_s} \log \left( Pr(\alpha_{ij}(t)) \right) - \sum_{i \in \mathcal{N}} \sum_{t=1}^{N_t} R_{ij}(t) \alpha_{ij}(t) \left( \mu^t_j P^R_j + \mu^t_i P^T_i \right) \Delta \\
- \sum_{i \in \mathcal{N}} \sum_{t=1}^{N_t} e_i(t)(\mu^t_i - \mu^{t+1}_i) + \sum_{i \in \mathcal{N}} \mu^1_i e_i(0) - \sum_{i \in \mathcal{N}} \sum_{t=1}^{N_t} \mu^t_i (C_i - S_i(t)) \Delta \\
= \sum_{l(i,j) \in f_s} \sum_{r_s} \left[ \log \left( Pr(\alpha_{ij}(t)) \right) - R_{ij}(t) \alpha_{ij}(t) \left( \mu^t_j P^R_j + \mu^t_i P^T_i \right) \Delta \right] \\
- \sum_{i \in \mathcal{N}} \sum_{t=1}^{N_t} e_i(t)(\mu^t_i - \mu^{t+1}_i) + \sum_{i \in \mathcal{N}} \mu^1_i e_i(0) - \sum_{t=1}^{N_t} \mu^t_i (C_i - S_i(t)) \Delta
\]

Therefore, the dual of the primal problem is: \[ \min_{\mu \geq 0} U(\mu), \] where the dual objective function \(U(\mu)\) is given as

\[
U(\mu) := \max_{\alpha \in A, \ e \in E} \ L(\alpha, e, \mu),
\]

\[
A := \{(\alpha_{ij}(t)) \mid \alpha_{ij}(t) \geq 1, \forall l(i,j) \in \mathcal{L}, 1 \leq t \leq N_t \},
\]

\[
E := \{(e_i(t)) \mid 0 \leq e_i(t) \leq B_i, e_i(N_t) \geq RE_i, \forall i \in \mathcal{N}, 1 \leq t \leq N_t \}.
\]

Furthermore, the dual objective is equivalent to the following two separate optimization problems:

\[
U(\mu) := U_1(\mu) + U_2(\mu) + \sum_{i \in \mathcal{N}} \left[ \mu^1_i e_i(0) - \sum_{t=1}^{N_t} \mu^t_i (C_i - S_i(t)) \Delta \right]
\]

where

\[
U_1(\mu) := \max_{\alpha \in A, \ e \in E} \sum_{l(i,j) \in f_s} \sum_{t=1}^{N_t} \left[ \log \left( Pr(\alpha_{ij}(t)) \right) - R_{ij}(t) \alpha_{ij}(t) \left( \mu^t_j P^R_j + \mu^t_i P^T_i \right) \Delta \right]
\]

\[
(3.14)
\]

and

\[
U_2(\mu) := \max_{\alpha \in A, \ e \in E} \sum_{i \in \mathcal{N}} \sum_{t=1}^{N_t} e_i(t)(\mu^{t+1}_i - \mu^t_i).
\]

Based on the above decomposition, we can implement SolarCode in a distributed way.
Iteratively, each node updates and exchanges its lagrangian multipliers with its neighbors, and solves $U_1(\mu)$ and $U_2(\mu)$ locally. Specifically, during the $\tau^{th}$ iteration, each node $i$ first broadcasts its prices $\mu^i_1(\tau)$ to all of its neighbors. Second, after receiving the updated prices, node $i$ solves the two optimization problems $U_1(\mu(\tau))$ (Eq (3.14)) and $U_2(\mu(\tau))$ (Eq (3.15)) locally, and determines its residual energy $e_i(t)$ and the replication factor $\alpha_{ij}(t)$ for each of its outgoing link $l(i, j)$, and then broadcasts $e_i(t)$ to all of its neighbors and unicasts $\alpha_{ij}(t)$ to each neighbor $j$. Finally, based on the current $\alpha_{ij}(t)$ and $e_i(t)$, each node $i$ updates its price $\mu^i_1$ by using the subgradient-based method:

$$
\mu^i_1(\tau + 1) = \left[ \mu^i_1(\tau) - h \frac{\partial U_1(\mu^i_1(\tau))}{\partial \mu^i_1} \right]^+ \\
= \left[ \mu^i_1(\tau) + h \left( e_i(t) - e_i(t - 1) - (S_i(t) - C_i - \sum_{l(i,j)} R_{ji}(t) \alpha_{ji}(t) P_{RX}^i - \sum_{l(i,j)} R_{ij}(t) \alpha_{ij}(t) P_{TX}^i \Delta) \right) \right]^+,
$$

where the step-size $h$ is a positive constant. It can be proved that the subgradient algorithm converges to within a range of the optimal value [56].

Suppose that transmitting one real number requires one message unit, the communication overhead of decentralized SolarCode, for one node in one iteration, consists of $N_t$ message units used to broadcast its $N_t$ lagrangian multipliers to its neighbors, $N_t$ messages to broadcast $e_i(t)$, and $N_tG$ messages units used to disseminate $\alpha_{ij}(t)$ to its $G$ neighbors. Let $I$ be the average number iterations that the algorithm needs to converge. Summing over all nodes and all iterations, the total message overhead is $N_t(\vert \mathcal{L} \vert + 2\vert \mathcal{N} \vert)I$. Comparing to the centralized solution that has an overhead of $[(N_t + 1)\vert \mathcal{L} \vert + \vert \mathcal{N} \vert]H$, the decentralized solution has advantage when $[(N_t + 1)\vert \mathcal{L} \vert + \vert \mathcal{N} \vert]H < N_t(\vert \mathcal{L} \vert + 2\vert \mathcal{N} \vert)I$. In fact, most deployments in practice including our solar-powered testbed has a relatively small $H$, while the number of iterations $I$ for convergence is large. Therefore, the centralized version is more appealing in real deployments.
3.4 Implementation Issues

While SolarCode is, in principle, applicable to a broad range of sensing systems with renewable energy sources, we discuss its implementation issues in the context of our solar-powered sensor network testbed as presented in Chapter 2. Moreover, the performance evaluation of SolarCode in Section 3.5 is also based on the real traces collected on this testbed.

SolarCode relies on the projection of the available solar energy. Many analytic, stochastic, empirical or more complex (e.g., artificial neural network based) models [57, 58, 59] have been proposed to estimate solar radiation on the earth horizon (note that solar radiation above the atmosphere is quite predictable). In this work, we utilize a simple empirical model, which is based on the historical data and weather forecast information. Historical data are used to find the correlations between the solar radiation and local weather parameters like sunshine duration, maximum temperature, and cloud cover. Thereby these correlations are used to project the solar radiation level based on the weather forecast information. Then the projected solar radiation level can be translated into the output current from a solar panel, given its rated power, pointing direction and the angle with the horizon. Figure 3.4 shows the real and projected solar energy traces during one day. The curve of the projected trace is smooth because only daily weather parameters are used. So the performance of the estimation depends on the accuracy of the weather forecast that can be obtained.

Due to the limitation in the weather forecast accuracy, we can expect errors between the projected and real traces, as shown in Figure 3.4. In order to enforce the non-blackout constraint, SolarCode can be tuned to follow a more strict residual energy lower bound such that the blackout probability is small. This bound can be determined based on the statistical properties of the estimation errors in the history.
3.5 Performance Evaluation

In this section, we evaluate SolarCode based on the real setting of our testbed. Figure 3.5 shows the topology of our 9 deployed nodes. In fact, all nodes are within the communication range of each other. However, only links with reasonably packet delivery probability ($\geq 30\%$) are picked in forming this topology in order to satisfy Theorem 2. Packet delivery probability of links between these nodes is measured on the testbed, and only links with packet delivery probability greater than 30% are selected to form routes between source and destination nodes. Table 3.1 lists the packet delivery probability $p_m$ for links in Figure 3.5. Totally 6 flows are used in our evaluation. Table 3.2 shows the flow routes and data rates in packets per second (100 bytes per packet). As discussed in Section 3.3, we implement the centralized SolarCode on the testbed because it has less message overhead.

Based on historical solar energy traces and weather forecast information, projected solar
Figure 3.6: Real (thin red curve) and projected (thick blue curve) solar energy traces during Oct 21st – Nov 4th 2008. Total energy collected daily in units of Ampere Hour is also shown: the numbers on the first line are for the real trace and the second are for the projected trace.

\[
\begin{array}{ccccccc}
\text{Link} & p_m & \text{Link} & p_m & \text{Link} & p_m & \text{Link} & p_m \\
(6,2) & .43 & (2,7) & .43 & (7,1) & .30 & (8,1) & .66 \\
(6,4) & .43 & (4,5) & .51 & (7,9) & .39 & (3,8) & .72 \\
(9,1) & .39 & (5,7) & .43 & (5,9) & .33 & (9,3) & .30 \\
\end{array}
\]

Table 3.1: Packet delivery probability for links used by at least one flow.

Energy input traces are generated as the input for SolarCode. Real solar energy traces are also collected on the testbed, and used as the input to compute the ground truth performance of SolarCode. Note that the solar energy traces on different nodes are very similar because our testbed is deployed in a half mile by half mile area where all nodes share almost the same solar radiation conditions. Hence only one pair of projected and real traces are shown in this chapter (Figure 3.6). Besides the charging current (in Amperes) from the solar panel over 15 days, the figure also shows the total amount of solar energy collected daily in units of

\[
\begin{array}{ccc}
\text{Flow Name} & \text{Data Rate (10^2 pkt/s)} & \text{Flow Route} \\
f_1 & 1.5 & 2-7-1 \\
f_2 & 0.5 & 5-7-1 \\
f_3 & 0.5 & 6-2-7-9 \\
f_4 & 1.25 & 4-5-9-3 \\
f_5 & 2.0 & 3-8-1 \\
f_6 & 1.0 & 6-4-5-9-1 \\
\end{array}
\]

Table 3.2: Data rate and route of 6 flows used in the evaluation.
Ampere Hour (AH), which equals the current flow in amperes multiplied by the time of the current in hours. As we can see, the projected and real daily solar energy match very well. The variance $\sigma^2$ of the estimation error is about 3.4 AH$^2$, and thus we use $3\sigma^2$ as the lower bound in the non-blackout constraint to avoid the blackout caused by estimation errors.

The replication factor schedule produced by SolarCode is not very intuitive to understand, especially when the traffic pattern is complex. Hence, we first show the experimental results of a basic scenario (flow $f_1$ and $f_2$ for 5 days), and then show the results of an extensive experiment (all 6 flows for 15 days). Finally we compare SolarCode with other code scheduling schemes.

Figure 3.7: The replication factor $\alpha$ (top), solar energy input (middle) and residual energy (bottom) for the experiment with flow $f_1$ and $f_2$ for 5 days.
3.5.1 SolarCode with Projected Solar Trace

We first study how SolarCode reacts to environmental (solar energy) changes, as well as differences in link quality and traffic load. In this experiment, we run flow $f_1$ and $f_2$ for 5 days. In Figure 3.7, the top part shows the replication factors scheduled by SolarCode on link (5, 7), (2, 1) and (2, 7); the middle part shows the projected solar energy trace for readers’ reference; and the bottom part shows the percentage of the residual energy with respect to the battery capacity on node 1, 2, 5 and 7. Four interesting observations are in order.

First, the replication factors $\alpha$ of all three links are adapted according to the energy level of sensor nodes throughout the 5 days. Normally, $\alpha$ is higher during daytime than during nighttime. But if weather conditions are not good (e.g., rainy in day 3 and day 4), $\alpha$ stays at a low level to save energy for avoiding blackout.

Second, among the three links, link (7, 1) has a higher $\alpha$ during the time with sufficient residual energy (day 1, 2 and 5) because it needs more redundancy than the other two links, which have better quality than (7, 1). One exception happens during the daytime of day 1, when link (5, 7) and (2, 7) have very high replication factors. This is due to the fact that all three nodes have plenty of residual energy for the first day by starting with a reasonably high initial energy. Moreover, the solar energy input in day 1 is also high. Hence there is lots of energy surplus for node 2 and 5 to spend as their batteries will be fully charged in day 1 anyway. But for days with low solar energy input, link (7, 1) gets a little bit lower $\alpha$ than the other two links. This is because saving energy now becomes more urgent than improving reliability and more energy can be saved by lowering the $\alpha$ of link (7, 1) due to its heavy traffic.

Third, link (2, 7) and (5, 7) have almost identical replication factors, even though the traffic carried on them is very different. Traffic on link (2, 7) and (5, 7) shares the receiving energy consumed on node 7. One may think that more energy or higher replication factor
Figure 3.8: The residual energy of node 1, 2, 5 and 7 when SolarCode is not used.

should be given to link (2, 7) because the data rate of flow $f_1$ on it is 3 times of the rate of flow $f_2$ on link (5, 7). The reason behind this is the energy consumption rate of a flow is also proportional to the flow rate. From the perspective of packets, they can reach the same reliability level by consuming the same amount of energy on the two links with the same quality. Thus there is no need to differentiate which flow these packets belong to. This actually can be formally proved based on Eq (3.11). Due to space limitation, we just give an informal explanation. Imagine that link (2, 7) is composed of 3 virtual links and each virtual link carries $1/3$ of flow $f_1$. Now all 3 virtual links and link (5, 7) are identical, and thus should be assigned the same replication factor. Thus the overall replication factor of the 3 virtual links (or the original link (2, 7)) is the same as that of link (5, 7).

Additionally, the actual residual energy of node 1, 2, 5 and 7 is obtained based on the real solar energy trace. As shown in the bottom part of Figure 3.7, SolarCode only spends energy surplus (if any) to enhance the delivery reliability and hence incurs no blackout. For comparison, we also show in Figure 3.8 the residual energy of the nodes when SolarCode is not used. We can see that part of the residual energy has never been utilized.

During the 5 days, the total number of packets delivered by the two flows is $5.56 \times 10^7$, resulting in an average end-to-end delivery probability of 63.4%. If SolarCode is not used, the average end-to-end deliver probability is only 12.9%. 

38
3.5.2 SolarCode with Real Solar Trace

If the exact solar energy trace were known as an oracle, we could expect that SolarCode could perform perfectly in allocating energy surplus. So we run the experiment again with the real solar energy trace, and show the replication factors of the 3 links in Figure 3.9.

Recall that SolarCode reserves $3\sigma^2$ energy to deal with errors in solar energy projection. Now with the exact energy trace as the input, SolarCode can allocate more energy for enhancing communication reliability. Comparing Figure 3.9 with Figure 3.7, $\alpha$s of all the three links are now a little bit higher and thus result in an average end-to-end delivery probability of 66.2%. We can see that SolarCode with projected traces has comparable performance to that with real traces.

3.5.3 An Extensive Experiment

In this experiment, we test SolarCode with all 6 flows for 15 days. For clarity, only the replication factors of interesting links are shown in Figure 3.10.

First, we observe again that the replication factor is regularly higher during daytime except in days of bad weather (e.g., day 3, 4, 7 and 13). Second, although we argued in the basic scenario that the replication factor is independent of the flow rate, this only holds when there is plenty of energy surplus on incident sensor nodes. In fact, it is not the case for most of the time. Consequently, the replication factor that a link could reach is limited by the energy availability on its incident nodes. Thus, if a node (e.g., node 5, 7 and 9) relays
Figure 3.10: The replication factor $\alpha$ by SolarCode for the experiment of all 6 flows for 15 days. Only 5 links are shown for the clarity of presentation.

Heavy traffic, the replication factors on its incident links (e.g., link (2, 7), (5, 7), (9, 3)) would usually be lower than those on links with low traffic (e.g., (8, 1) and (6, 4)). Third, also because of heavier traffic on link (2, 7) and (5, 7) than in the basic scenario, the replication factors on these links are lower than in the basic scenario.

During the 15 days, the average end-to-end delivery probability is 89.9% by using SolarCode, much higher than the delivery probability 19.1% when no SolarCode is used. And it is comparable to the delivery probability 90.7% achieved by SolarCode with real solar traces.

3.5.4 Comparison to Other Schemes

Besides SolarCode and the no coding scheme, one intuitive method is to use the erasure coding only when the battery is full. We call this method FullCode, which obviously has no impact on the network lifetime. Comparing to SolarCode, one advantage of FullCode is that it does not require the projection of solar energy. Basically, it can just monitor the online status of the battery and charging current from the solar panel, and start the erasure coding when the battery is full and a charge current is still available. As mentioned before, it is unwise to spend all energy surplus on one link because only little additional reliability could
Figure 3.11: The replication factor $\alpha$ by FullCode, which uses erasure codes only when the battery is full.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$f_1$ and $f_2$</th>
<th>all 6 flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Coding</td>
<td>12.9%</td>
<td>19.1%</td>
</tr>
<tr>
<td>SolarCode with projected traces</td>
<td>63.4%</td>
<td>89.9%</td>
</tr>
<tr>
<td>SolarCode with real traces</td>
<td>66.2%</td>
<td>90.7%</td>
</tr>
<tr>
<td>FullCode</td>
<td>20.4%</td>
<td>34.1%</td>
</tr>
</tbody>
</table>

Table 3.3: Average end-to-end packet delivery probability of different schemes in each of the two scenarios.

be gained when the redundancy level of that link is already high. Therefore, FullCode has an upper bound on the replication factor of each link, which is the factor that can enhance the link delivery probability to 99.9%. This bound can be calculated for each link based on its link quality and the characteristics (e.g., block number $b$) of erasure codes.

We run FullCode with the real solar energy trace under each of the two scenarios. The replication factor in the basic scenario are shown in Figure 3.11. We can see that FullCode spends energy surplus in a more conservative way. Consequently, as summarized in Table 3.3, the average end-to-end packet delivery probability achieved by FullCode is much lower than that achieved by SolarCode.

### 3.6 Conclusion

In this chapter, we consider the problem of utilizing energy surplus in solar-powered sensor networks to adaptively adjust the redundancy level of erasure codes, such that the end-to-end packet delivery probability is improved and the network lifetime is conserved. We
formulate this as an optimization problem. Because of its intractability, we propose an approximated solution called SolarCode, and prove that it only has a constant approximation ratio. We evaluate SolarCode in the setting of our solar-powered testbed. Results show SolarCode schedules the redundancy level of each link dynamically according to the solar energy harvesting process. As a result, the end-to-end packet delivery probability is increased to 89.9%, comparing to 19.1% when no SolarCode is used, and 34.1% when using a naive code scheduling scheme.
Chapter 4

Reliable In-network Data Storage

In this chapter, we solve the problem of providing adaptive storage reliability in solar-powered, remotely-deployed sensing systems. We consider collected data as the main output to be protected, hence measuring reliability in terms of the amount (or utility) of successfully recoverable data.

4.1 Introduction

In many remotely-deployed systems, sensor nodes have no permanent connectivity to the outside world, and thus sensory data have to be stored in the network until the next upload opportunity (e.g., a mobile basestation) appears. Reliable delivery of all sensory data that are cached in the network may not be always possible because of the following two factors. First, data stored in the network may be subject to uncontrolled loss due to hardware failures that are caused by adverse natural events, such as fire or floods, in outdoor harsh environments. For example, one node in our sensing testbed was damaged due to a short-circuit caused by heavy rainfall and flooding (shown in Figure 4.1). Second, the dynamic nature of solar energy may also cause uncontrolled energy depletion, which results in loss of sensing ability.

In this chapter, we adopt a predominantly disconnected network model, and propose a series of solutions to maximize retrievable data in the face of node failures and variations in available energy supply. Reliability can be achieved using redundancy. In stead of using simple data duplication, as in Chapter 3, we utilize erasure codes for data replication because
of its higher recovery efficiency. Specifically, sensory data are organized as chunks, and each data chunk is partitioned into $b$ blocks and encoded into $b'$ ($b' \geq b$) encoded blocks. Then, the encoded blocks are distributed onto the nodes of the system. The original data chunk can be recovered as long as any $b$ out of the $b'$ encoded blocks are retrieved. By working on data blocks with finer granularity, erasure coding can achieve higher reliability than simple replication with the same energy and storage requirements. For example, if $b = 8$ and $b' = 12$, the original data can be recovered when any 8 out of 12 blocks survive. This means that erasure codes with only $12/8$ redundancy can tolerate up to 4 failed nodes, while simple data duplication needs 5 replicas.

Maximization of retrievable data implies minimizing loss due to node damage in harsh environmental conditions, as well as minimizing sensing blackouts due to energy depletion. Following the formulation of SolarCode in Chapter 3, one can formulate an optimization problem to maximize the total expected amount of data that can be recovered under a node failure model while ensuring none energy blackout. However, solving this optimization requires knowledge of both the node failure probability distribution (to predict replication needs) and weather patterns (to predict energy input). Therefore, in this chapter, we will first explore another solution direction, that requires no prior knowledge of these physical models.
A key insight that we discovered is that the energy status of a node is reset every time the battery is fully charged. At that time, prior energy use becomes irrelevant. Instead, based on the average energy charge and discharge rates, we can determine how much energy should be reserved in order to sustain the node until the next time it is fully charged. The energy surplus (if any) can be used to improve storage reliability. Based on this insight, we propose a scheme, called **SolarStore**, which dynamically determines whether energy surplus exists or not, and turn on and off the data replication process based on the availability of energy surplus.

To a further extent, we consider the problem of reliable data collection in a multi-criticality scenario, providing different levels of Quality of Storage (QoS) for data of different criticality/utility values. Applications on sensing systems may involve heterogeneous sets of sensors. For instance, applications to monitor climate change may require a diverse mixture of temperature, pressure and humidity sensors. The sensory data from different sensors or even different readings from the same sensor may have different interest/utility to the end users. For example, an application may be more interested in temperature changes or more interested in abnormal events as opposed to normal conditions. Hence, there arises the need for multi-criticality data support, which allows data with a higher utility to have a higher priority in using energy and storage resources in order to enhance overall storage reliability under resource constraints. For the sake of diversity, unlike SolarStore, we formulate this problem in a more rigorous way, by assuming the knowledge of node failure models and weather patterns, and propose a solution called **SolarQoS**.

The massive data collected on these high-end sensing systems can run out the system storage from time to time. Thus, in some situations, data have to be dropped in a discriminative manner because not all data are equally important. Moreover, instead of being static, the utility of data can also dynamically depend on the data that have been collected before. For example, pictures for a point of interest from different angles carry more information than pictures from similar angles. Therefore, we need to allocate the scarce resources based
on the degree of similarity (or dissimilarity) of the data. To this end, we propose a storage replacement policy, called SimStore. It extracts features from each piece of data (e.g., sensory data chunk), dynamically determines the utility of the data based on its similarity with other data collected, and then discards the data so that the diversity of the remaining data is maximized.

The remainder of the chapter is organized as follows. In Section 4.2, we present the simple but effective solution SolarCode. In Section 4.3, we propose SolarQos, taking into account of a static utility model. In Section 4.4, we present our content-aware data replacement policy SimStore based on the dynamic similarity of data.

4.2 SolarStore: A Threshold-based Storage Replication Service

In this section, we formulate the reliable in-network data storage as an optimization problem, and propose a simple and effective solution, called SolarStore, which does not require prior knowledge of the physical models on node failure rates and weather patterns. It dynamically maintains an energy-surplus threshold, and uses the residual energy above this threshold to replicate and disseminate the sensory data to improve storage reliability.

4.2.1 Problem Formulation

In storage-centric sensor networks, there are two types of possible reasons for data loss. One is loss due to node failures (e.g., caused by natural events in the harsh environment). The data on failed nodes cannot be retrieved. Redundancy can be used to improve storage reliability and thus mitigate data loss caused by node failure. Achieving redundancy takes additional energy and storage resources to replicate/encode data and scatter them in the network. Let $E_{\text{residual}}$ be the current residual energy in the battery, and $\Delta E (0 \leq \Delta E \leq E_{\text{residual}})$ be the
energy allocated for enhancing data reliability. Given a replication method, the status of the
current buffered data and the node failure model, we can determine how much more data
can survive if $\Delta E$ is spent, and denote this gain in data reliability by $Gain(\Delta E)$.

On the other hand, it is possible that the added energy consumption will lead to suspension
sensing due to energy depletion and hence result in data loss. Denote $Loss(\Delta E)$ as the
additional amount of such data loss if $\Delta E$ amount of energy is taken away for data reliability.
Our goal is

$$\max \quad Gain(\Delta E) - Loss(\Delta E) \quad (4.1)$$
$$s.t. \quad 0 \leq \Delta E \leq E_{\text{residual}}.$$

Intuitively, the larger the $\Delta E$, the higher the $Gain(\Delta E)$ in data reliability but potentially
the more the $Loss(\Delta E)$. Hence $\Delta E$ should be carefully chosen such that $Gain(\Delta E)$ and
$Loss(\Delta E)$ are balanced.

$Loss(\Delta E)$ can be further quantified using the insight that the energy in the system is
renewed every time the battery is fully charged. It implies that it does not matter how
the energy was spent before, once the battery is fully charged again. Therefore, given the
current residual energy $E_{\text{residual}}$, we define $B(E_{\text{residual}})$ as the expected duration of blackout time
(i.e., when the battery is empty and data sensing is suspended), from now until the
next time when the battery is fully charged. Since no sensory data can be created during
a blackout, the resulting data loss is $R \cdot B(E_{\text{residual}})$, where $R$ is the expected data sensing
rate. Thus, if $\Delta E$ is allocated from $E_{\text{residual}}$ for storage reliability, we have the additional
data loss

$$Loss(\Delta E) = R \cdot [B(E_{\text{residual}} - \Delta E) - B(E_{\text{residual}})] \quad (4.2)$$

The explicit formula for $B(E_{\text{residual}})$ and $Gain(\Delta E)$ depends on physical models used for
node failure and weather patterns. The question becomes whether we can identify some
properties of $Gain(\Delta E)$ and $B(E_{\text{residual}})$ that are independent of those physical models and
can help in solving the problem. In the next subsection, we describe a solution to this problem.

### 4.2.2 A Dynamic Threshold-based Solution

We propose a simple and effective solution based on the following properties of \( \text{Gain}(\Delta E) \) and \( B(E_{\text{residual}}) \).

**Property 1**: Given any node failure model, \( \text{Gain}(\Delta E) \) is a non-decreasing function of \( \Delta E \). This can be ensured by any correct reliable storage scheme, which increases or at least maintains the current reliability level if more energy is spent.

**Property 2**: Given any weather pattern, \( B(E_{\text{residual}}) \) is a non-increasing function of \( E \) under the same power consumption profile. It is obvious that the higher the energy available initially, the lower the blackout time, under the same weather conditions and the same power consumption profile.

Let \( P_{\text{solar}} \) be the average power charging rate of the solar panels, and \( P_{\text{sys}} \) be the average power consumption rate of the system. Given the current residual energy \( E_{\text{residual}} \), the expected number of days to the next time when the battery is full can be calculated by

\[
T_{\text{full}}(E_{\text{residual}}) = \frac{C - E_{\text{residual}}}{P_{\text{solar}} - P_{\text{sys}}},
\]

where \( C \) is the battery capacity.

Note that when the battery is partially empty, to charge it, it must be that \( P_{\text{solar}} \geq P_{\text{sys}} \), which means that the average energy consumption rate of the system must be made less than or equal to the average solar energy charging rate. Otherwise, the system will shut down.

Even though solar energy varies from daytime to night-time and from day to day, it remains true that the expected blackout time is zero between now and the next time the
battery is full, if we currently have at least $P_{sys} \cdot T_{full}(E_{residual})$ energy in the battery. This is true even in the worst case, where all solar energy charging occurs at the very last instant at $T_{full}$. By solving $E_{residual} = P_{sys} \cdot T_{full}(E_{residual})$, we have $E_{residual} = \frac{P_{sys}}{P_{solar}} C$. Based on the monotonicity of $B(E_{residual})$ as in Property 2, we obtain the last but most important property.

**Property 3:** $B(E_{residual}) = 0$ if $E_{residual} \geq \frac{P_{sys}}{P_{solar}} C$. It implies that by maintaining the residual energy above a fraction $\frac{P_{sys}}{P_{solar}}$ of the battery capacity $C$, the expected data loss due to blackout is always zero.

Based on Property 3, when the energy allocation $\Delta E$ for storage reliability satisfies $E_{residual} - \Delta E \geq \frac{P_{sys}}{P_{solar}} C$, we have

$$Loss(\Delta E) = R \cdot B(E_{residual} - \Delta E) - R \cdot B(E_{residual}) = 0 - 0 = 0. \quad (4.4)$$

Furthermore, according to Property 1 that the reliability gain $Gain(\Delta E)$ is a non-decreasing function of $\Delta E$, we thereby allocate

$$\Delta E = E_{residual} - \frac{P_{sys}}{P_{solar}} C \quad (4.5)$$

for SolarStore to maximize the reliability gain $Gain(\Delta E)$, at the same time, with no additional data loss in data sensing (i.e., $Loss(\Delta E) = 0$).

Following this allocation method, when the residual energy $E_{residual}$ is less than the threshold $\frac{P_{sys}}{P_{solar}} C$, all energy will be reserved for data sensing. Only when the residual energy is above the threshold, the *energy surplus* ($\Delta E = E_{residual} - \frac{P_{sys}}{P_{solar}} C$) can be used for enhancing data reliability, at no expected additional cost in data sensing.

This allocation method requires the knowledge of $P_{sys}$ and $P_{solar}$, which can be estimated online by using moving averages. Let $P_{sys}^{new}$ and $P_{solar}^{new}$ be the latest samples for the system.
power consumption rate and solar panel charging rate respectively. The moving averages can be computed by:

\[ P_{sys} = (1 - \theta_{sys})P_{sys} + \theta_{sys}P_{sys}^{new}, \quad (4.6) \]

\[ P_{solar} = (1 - \theta_{solar})P_{solar} + \theta_{solar}P_{solar}^{new}, \quad (4.7) \]

where the forgetting factor \( \theta_{sys} \) (0 < \( \theta_{sys} \) < 1) and \( \theta_{solar} \) (0 < \( \theta_{sys} \) < 1) controls how fast the historical samples should be neglected. How to choose \( \theta_{sys} \) and \( \theta_{solar} \) will be addressed in Section 4.2.4, together with other implementation issues for SolarStore. Moreover, if a duty cycle scheduling mechanism [31, 32] is employed to further save energy in data sensing, its affect will be captured by the calculation of \( P_{sys} \) and then reflected during the energy allocation.

We emphasize that this solution is simple and general, independent of physical models on node failure and weather patterns. Although this solution may not be optimal, it can guarantee that the system performance is never degraded (i.e., \( Gain(\Delta E) - Loss(\Delta E) \) is always non-negative).

Similarly, the storage resource is renewed every time when data are uploaded from the network. Thus, storage allocation can be performed in a similar way as the energy allocation. Denote \( M \) as the expected time from now to the next upload opportunity. Then, a storage space of \( R \cdot M \) is needed to store the future sensory data, where \( R \) is the expected data sensing rate. Let \( S_{\text{residual}} \) be the current residual storage space left. When \( S_{\text{residual}} \) is above \( R \cdot M \), the storage surplus

\[ \Delta S = S_{\text{residual}} - R \cdot M \quad (4.8) \]

can be allocated for storing data replicas.
4.2.3 Data Encoding and Dissemination Design

Instead of using simple replication of data chunks, we use erasure coding for data replication. Specifically, sensory data are organized as chunks, and each data chuck is partitioned into \( b \) blocks and encoded into \( b' \) (\( b' \geq b \)) encoded blocks. Then, the encoded blocks are distributed onto the nodes of the system. The original data chunk can be recovered as long as any \( b \) out of the \( b' \) encoded blocks are retrieved. In this section, for the sake of simplicity, we assume that \( b \) and \( b' \) are fixed parameters in erasure coding. Then three questions arise in this context.

The first one is where to scatter the encoded data blocks. The ideal scenario is to distribute them evenly among all nodes in the network. However, this requires that each node have a global view of the network. Maintaining such a view is quite expensive because the network is dynamic as nodes could die or come alive later. Instead, we adopt a very simple heuristic in which nodes only scatter encoded blocks to their neighboring nodes. As will be explained later, those blocks could be further forwarded away by the neighbors if their energy permits.

The second question is how many encoded blocks should be scattered out to each neighbor. Recall that \( b' \) is the number of encoded blocks that are generated from each data chunk. Suppose a node has \( g \) neighbors and \( g + 1 \leq b' \), we scatter the blocks evenly so that there are \( b'/(g + 1) \) blocks on every neighbor and the node itself. If \( g + 1 > b' \), \( b' - 1 \) nodes are randomly picked from \( g \) neighbors, and one block is scattered to each of the \( b' - 1 \) selected nodes and the last block is kept on the current node itself. However, the neighborhood of a node is dynamic as data dissemination on some of neighboring nodes might be suspended or some neighboring nodes could fail completely. Therefore, we introduce a mechanism to allow encoded blocks to be redistributed again.

Hence, there comes the last question of when to redistribute those encoded blocks that were generated and kept on a node in the first place or received from other nodes. On each node, for each data chunk, let \( h \) be the number of encoded blocks stored on the node that
were generated from this data block. We define the reliability level for a data block on a	node as $\rho = b'/h$ \(^1\), which is the total number of encoded blocks generated for this data
chunk over the number of encoded blocks stored on this node for this data chunk. Given a
constant $b'$ used by erasure codes, the lower the $h$ is, the fewer encoded blocks are stored on
this node, and thus the more reliable is the original data chunk with regard to the possible
failure of this node. Note that, data chunks that have not been encoded by erasure codes
yet have the lowest reliability level $\rho = 1$ since $b' = b = 1$ for them. Based on the reliability
level $\rho$ of each data chunk on a node, the averaged reliability $\bar{\rho}$ for all data stored on this
node can be calculated.

We sort the data chunks according to their reliability levels, and select the data chunk
with the lowest reliability level to process first. If the data chunk has not been encoded yet,
then we encode it and scatter the encoded blocks evenly to neighbors as described before. If
the data chunk has been encoded before, then we randomly pick $b'(1/\rho - 1/\bar{\rho})$ encoded blocks
from its $h$ stored blocks, and scatter them evenly to neighbors. Hence, the reliability level
of this data chunk is increased to the average reliability level $\bar{\rho}$. In order to avoid sending a
block back to the nodes where it has been stored before, a list of nodes where this block has
been stored is maintained in each block, and nodes on the list are excluded when scattering
this block further.

If there is energy surplus $\Delta E$ and storage surplus $\Delta S$, a node starts to listen on a specified
port for incoming encoded blocks from other nodes. When a block is received, we first retrieve
the ID of its corresponding data chunk from the meta information in its header, and then
store it together with other blocks generated from the same data chunk.

### 4.2.4 Implementation Issues

SolarStore provides an adaptive reliable storage service to applications based on the status
of energy and storage resources. Chunks of data are passed by applications to SolarStore

\(^1\)The reliability level $\rho$ is slightly different from the replication factor $\alpha$ defined as $b'/b$. 

52
through the defined APIs, and then stored into a Repository, which is a piece of storage space on the solid state disk managed by the operating system.

A Replicator reads data chunks from Repository and encodes them into encoded blocks by using erasure codes, and then scatters the encoded blocks as described in Section 4.2.3. A Receiver receives the encoded blocks from other nodes and stores them into Repository, organizing blocks generated from the same data chunk together in one directory.

A Resource Allocator (as summarized by the pseudo code in Algorithm 2) monitors the status of energy and storage resources and determines the energy and storage surpluses, based on the solution proposed in Section 4.2.2. Replicator is stared when there is energy surplus because it demands energy to encode the data chunks into blocks and scatter them to other nodes, while Receiver is only started when both energy and storage surplus are positive as it requires both energy to receive and storage to store encoded blocks from other nodes. The scattering of encoded blocks by Replicator on a node also relies on Receiver on other nodes. One may suggest that Replicator could go sleep for a while if it found no live Receiver on other nodes. However, during its sleeping, it may miss the working period of Receiver of others. As a matter of fact, this idea deviates from the design principle of solar-
Algorithm 2 Resource Allocator

1: while true do
2: Sample the status of $E_{\text{residual}}$, $P_{\text{sys}}^{\text{new}}$, $P_{\text{solar}}^{\text{new}}$, and $S_{\text{residual}}$.
3: Update $P_{\text{sys}} = (1 - \theta_{\text{sys}})P_{\text{sys}} + \theta_{\text{sys}}P_{\text{sys}}^{\text{new}}$.
4: and $P_{\text{solar}} = (1 - \theta_{\text{solar}})P_{\text{solar}} + \theta_{\text{solar}}P_{\text{solar}}^{\text{new}}$.
5: Compute energy surplus $\Delta E = E_{\text{residual}} - \frac{P_{\text{sys}}}{P_{\text{solar}}}C$.
6: and $\Delta S = S_{\text{residual}} - R \cdot M$.
7: if $\Delta E > 0$ then
8: Start Replicator
9: else
10: Stop Replicator
11: end if
12: if $\Delta E > 0$ and $\Delta S > 0$ then
13: Start Receiver
14: else
15: Stop Receiver
16: end if
17: end while

powered systems and the definition of energy surplus, where nodes should be encouraged
to spend their energy surplus because of the renewable feature of solar energy. Therefore,
Replicator and Receiver keep running until energy surplus or storage surplus is used up.
Furthermore, as will be seen in Section 4.2.5, the behavior of SolarStore on nodes even with
different initial states will tend to become cooperative, which then leads to more overlapped
working periods of Replicator and Receiver on different nodes.

One issue in implementing SolarStore on our testbed is to choose the forgetting factors
$\theta_{\text{solar}}$ and $\theta_{\text{sys}}$ to calculate the moving average of the charging rate $P_{\text{solar}}$ and discharging
rate $P_{\text{sys}}$. Since SolarStore aims at perpetual systems that operate for a really long time, the
forgetting factors should be small so that the historical samples influence the averages for a
long period. Suppose we like the sample at $H$ time ago to have a weight of $\psi$ in the moving
average, then the forgetting factor $\theta$ should satisfies $(1 - \theta)^{H\lambda} = \psi$, namely $\theta = 1 - \psi^{1/H\lambda}$,
where $\lambda$ is the sampling rate. Considering that solar energy varies a lot from daytime to
night, $H$ should be at least one day. Furthermore, impact of weather changes in a short
period of time should also be smoothed out. On the other side, $H$ could not be too large in
order to respond to slight changes in the climate and seasons. To this end, we use $H = 5$ days and $\psi = 0.1$ in the current implementation.

Choosing parameters $b$ and $b'$ for data replication could also be tactical. The smaller $b$, the finer granularity of the replication, and thus the more reliability levels available. On the other hand, shifting encoded data blocks to other nodes is in units of blocks, which could lead to higher transmission overhead if blocks are too small because of a large $b$. Therefore $b$ should be carefully selected to trade off between replication granularity and transmission overhead. Meanwhile, the higher the $b'$ is, the higher the reliability level that can be achieved. This is also limited by the total available storage space $S_{\text{max}}$. Note that, upload opportunities appear with an expected period of $M$ and the expect data sensing rate is $R$, thus $b'$ should satisfy $b' \leq \frac{S_{\text{max}}}{RM}$ in order to mitigate data loss in data sensing. To this end, we use $b = 8$ and $b' = 12$ in the current implementation.

4.2.5 Performance Evaluation

In our performance evaluation, we consider a sensing application that collects bird vocalizations at 220.5 kHz and 16 bits per sample for environmental studies of bird populations and social behavior. This, in fact, is the real application that motivated the outdoor deployment. The data are being used by colleagues in the department of natural history at the University of Illinois at Urbana-Champaign. We run each experiment for a period of 15 days. As in Chapter 3 (Figure 3.6), the same trace from the outdoor testbed during Oct 21st – Nov 4th is used to emulate the solar energy input for the nodes indoors. Note that the solar energy harvesting traces on different nodes are very similar because the testbed is deployed in a farm-wide area where all nodes share almost the same weather conditions.

During the experiments, the application on a node always runs to record acoustic data unless the node is shut down when its battery is empty. The emulated battery of each node has a capacity of 98 AH; the same as the DEKA 8G31 battery used outdoors. Each node has a 18 GB space on its solid state disk to buffer the sensory data. A data mule arrives
every 3 days to collect all the recorded acoustic data from every node. Considering the large amount of data generated by the application, the data mule uses USB cables, instead of wireless communication, to connect to the nodes to copy the data. Thus, the time and energy used to upload the data from each node to the data mule are negligible.

Next, using the solar energy traces collected locally, we study the behavior of SolarStore under different energy conditions. Then, we show that SolarStore can also adapt to other (more extreme) environmental conditions. Finally, we compare SolarStore to three other schemes under different node failure scenarios.

A. Behavior study under different energy states

In order to study SolarStore under different energy conditions, we start each experiment with nodes having different initial energy in their batteries, evenly distributed between 10% to 90% of the battery capacity.

Figure 4.3 demonstrates how the residual energy $E_{\text{residual}}$ and the threshold $\frac{P_{\text{sys}}}{P_{\text{solar}}} C$ vary under SolarStore for 2 nodes with very different initial amount of energy. For your reference, the charging current from solar panels is also shown in the figure. For the sake of presentation,
energy surplus $\Delta E$ is not plotted but can be inferred by $\Delta E = E_{\text{residual}} - \frac{P_{\text{sys}}}{P_{\text{solar}}}C$. In the experiments, node 2 starts with a very low energy in its battery. For the first 9 days, its residual energy $E_{\text{residual}}$ is always below the threshold $\frac{P_{\text{sys}}}{P_{\text{solar}}}C$, and thus SolarStore allocates no energy for enhancing data reliability but reserves all energy for data sensing. When $E_{\text{residual}}$ eventually reaches above $\frac{P_{\text{sys}}}{P_{\text{solar}}}C$ on the 10th day, Replicator and Receiver (having storage surplus as well) begin to work. Then the energy is consumed in a faster rate, and Replicator and Receiver stops working after a few hours when $E_{\text{residual}}$ falls below $\frac{P_{\text{sys}}}{P_{\text{solar}}}C$ again. In each of the following days, there are always a few hours when $E_{\text{residual}} \geq \frac{P_{\text{sys}}}{P_{\text{solar}}}C$, and thus energy is allocated for improving data reliability during these periods of time. SolarStore on node 9 has similar behavior, except that there is energy surplus in the first two days because node 9 starts with an almost fully charged battery.

Comparing SolarStore on node 2 and node 9, even though they behave differently in the beginning because of the different initial states, the residual energy and the threshold trend to converge to some extent after a few days. Therefore, from the perspective of a perpetual system, the behavior of SolarStore in a long run does not depend on the initial state.

Next, Figure 4.4 illustrates the changes in the residual storage space $S_{\text{residual}}$ and storage surplus $\Delta S$ of node 2 and node 9 over the 15 days. A data mule comes every 3 days to collect all the data. Thus, the storage is renewed once every 3 days. As in Figure 4.4(b), during the first two days, nodes 9 has a storage surplus, and also has an energy surplus for most time as shown in Figure 4.3 (b). Thus its Replicator and Receiver start working, and then the storage surplus is gradually consumed by data replicas. In the next 3 days, the storage surplus remains relatively constant since Replicator and Receiver are sleeping during this period due to the shortage of energy as shown in Figure 4.3 (b). Starting from the 8th day, Replicator and Receiver wakes up to work for a few hours everyday. As shown in Figure 4.3 (a), The situation on node 2 is similar to node 9, except that the time when Replicator and Receiver starts working is later since the initial $E_{\text{residual}}$ of node 2 is lower. Note that an increase of $\Delta S$ could happen when encoded blocks sent out by Replicator are more than
Figure 4.4: Changes of the residual storage space $S_{\text{residual}}$ and storage surplus $\Delta S$. Those received by $\text{Receiver}$. And how to coordinate energy sharing between $\text{Replicator}$ and $\text{Receiver}$ will be addressed in our subsequent work.

Figure 4.5 shows the average reliability level $\bar{\rho}$ of the data on node 9 over the 15 days. Recall that the reliability level for raw data is 1. Several observations are in order. First, in the beginning of the experiment, node 9 starts $\text{Replicator}$ and $\text{Receiver}$, and then $\bar{\rho}$ soars from 1 to over 11. The reason of this steep increase is that only little data has been collected and stored locally so far. As we can see, this also happens every time after the data are retrieved by the data mule. Second, from day 3 to day 8, $\bar{\rho}$ remains one since $\text{Replicator}$ and $\text{Receiver}$ are turned off due to negative $\Delta E$. Third, after day 9, $\bar{\rho}$ is roughly controlled between 2 and 4, and trends to converge as the time goes on.

The average reliability level $\bar{\rho}$ of all 9 nodes after 15 days is shown in Table 4.1. Even though nodes starts with very different energy states, the achieved reliability levels on the 9 nodes are comparable, falling between 2.9 and 4.2. On average, the data reliability is about 3.56, which means that there are averaged $b'/3.51 = 3.37$ encoded blocks for each data chunk stored on a node. Thus $b' - 3.42 = 8.63$ blocks will still be retrievable even if one node is
Figure 4.5: Average reliability level $\bar{\rho}$ of the data on node 9.

<table>
<thead>
<tr>
<th>Node</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bar{\rho}$</td>
<td>3.3</td>
<td>4.1</td>
<td>3.2</td>
<td>3.4</td>
<td>2.9</td>
<td>3.6</td>
<td>4.2</td>
<td>3.9</td>
<td>3.4</td>
</tr>
</tbody>
</table>

Table 4.1: Node Average Reliability Level.

completely dead. According to fountain coding, the original data blocks are still recoverable as $8.63 \geq b$.

B. Adaptation to other environments

In order to study how SolarStore adapts to other environments, especially under some extreme environmental conditions, we emulate an extreme solar energy input based on the one used in the previous experiment. We enlarge the charging current by 3 times for one day every three days. And for the other two days in each cycle, we multiply it by a factor of 0.2. This way, solar energy input for SolarStore is highly skewed. We start the experiment with a fully charged battery for every node.

In Figure 4.6, we see that SolarStore adjusts the threshold $\frac{P_{sys}}{P_{solar}} C$ according to the solar input, increasing it slowly during the days of poor weather and decreasing it quickly to a reasonable level when a large amount of energy is charged into the battery. Figure 4.7 shows how the residual storage and the storage surplus varies during the experiment. The
Figure 4.6: Residual energy $E_{\text{residual}}$ and the threshold $\frac{P_{\text{sys}}}{P_{\text{solar}}} C$ of node 9 under a highly skewed solar energy input.

storage surplus remain roughly the same during days of bad weather, and is used up quickly during those extremely “sunny” days because during that time nodes are encouraged to spend energy on enhancing data reliability.

C. Comparison to three other schemes

Next, we evaluate the performance of SolarStore in enhancing data reliability, comparing to three other schemes: (1) $0$-Reliable has no data replication at all and uses all energy and storage space for data sensing; (2) $1$-Reliable always replicates data to maximize data reliability; and (3) full-Reliable only starts data replication when the battery is nearly full (99%) because the energy charged from solar panels will be wasted if not used. We conduct the experiments under three node failure scenarios, in each of which 1, 2, or 3 nodes are randomly chosen to be dead right at the end of each experiment. Recall that there are two types of data loss. One is in data sensing during energy blackout, while the other one is the data that are on failed nodes and cannot be recovered as no enough encoded blocks are found on other working nodes. All experiments are conducted under the same solar energy conditions as shown earlier in Figure 3.6.
Figure 4.7: Changes of the residual storage space $S_{\text{residual}}$ and storage surplus $\Delta S$ of node 9 under a highly skewed solar energy input.

Figure 4.8 shows the total data loss of the four different storage schemes under the 3 different node failure scenarios. In the figure, results are first grouped based on the number of failed nodes. Then in each group, 4 bars represent the total data loss for the 4 schemes as labeled. The data loss in each case further breaks down into the two types. As we can see, SolarStore has the lowest total data loss in all cases. Even though $1$-Reliable achieves the best in recovering data from node failures, its overall performance is lower because of the severe data loss during energy blackouts. On the contrary, $0$-Reliable has no replication and recovery mechanism, and thus has the worst data loss caused by node failures. Even though $\text{full}$-Reliable improves the data reliability to some extent, it still suffers at least 58% more data loss comparing to SolarStore, since it is too conservative in energy allocation for data reliability. This experiment justifies our design and illustrates the efficacy of our mechanisms.

D. Discussion

The proposed SolarStore is a simple and effective solution, and it is independent of physical models of node failure and weather patterns. However, it might not be optimal, and thus
can be enhanced by incorporating more sophisticated models of node failure and weather patterns. Moreover, SolarStore regards all data of the same utility to end users, but this may not be the case for many applications. Therefore, another avenue for enhancing SolarStore is to introduce multiple levels of criticality for sensory data, and allow data with higher criticality to have higher priority in utilizing system resources. We will investigate this in the next Section.

4.3 SolarQoS: Providing Quality of Storage based on Static Utility

To improve storage reliability against hardware failures, in this section, we propose an adaptive storage service, called SolarQoS, that provides different levels of Quality of Storage (QoS) for data of different utility values.

4.3.1 Problem Formulation

We consider a sensing system with a node set $\mathcal{N}$, and write $N = |\mathcal{N}|$ to denote the system size. Sensory data are collected on nodes and then stored in the network until the next upload.
opportunity appears. Each sensor node is equipped with a set of heterogenous sensing devices to sense the physical world. As we mentioned, data from different sensors or even different sensor readings from the same sensor may have different utility to end users. In this work, we classify the sensory data into categories according to their utility values. Formally, let $\mathcal{K}$ be the set of all utility categories of the data to be delivered to end users. For each category $k \in \mathcal{K}$, the end users specify a value $U^k$ as its interest/utility. Moreover, different nodes may have different data input rates because of an uneven distribution of the events of interest. So we denote as $R^k_i$ the input rate of the data in utility category $k$ on node $i \in \mathcal{N}$.

We use erasure codes replicate data for storage reliability against node failures. Sensory data are organized into chunks. Each data chunk is first divided into $b$ blocks, and then $\alpha b$ encoded blocks are generated by using erasure codes. Recall that the parameter $\alpha$ determines the degree of redundancy and is called the replication factor. When a node fails, we consider the worst case that all data on this node are lost. The corruptions of encoded blocks could be correlated especially when they are on the same node. From a global perspective, however, each block has an independent corruption probability considering that all the encoded blocks are indistinguishable in terms of data recovery and they are independently and randomly scattered among all the nodes. Thus, rather than depending on which nodes fail and whether the failures are correlated, the corruption probability of each block is just the probability that this block is stored on one of the failed nodes, which is $\frac{m}{N}$ where $m$ is the number of failed nodes.

Therefore, we are able to incorporate a very general failure model, where node failures can be either independent or correlated. The only thing that the model needs to specify is the probability that failures happen to $m$ out of $N$ nodes, denoted as $P_1(m)$. For example, if nodes fail independently with a probability $p$, we have $P_1(m) = \binom{N}{m} p^m (1-p)^{N-m}$. We assume that $P_1(m)$ for $0 \leq m \leq N$ is obtained through empirical studies and used here as an input of this problem.

When $m$ nodes fail, the probability that a data chunk can survive equals to the probability
that at least $b$ out of all its $\alpha b$ encoded blocks are stored on the $N - m$ alive nodes. We can formally express this recovery probability as

$$
\widetilde{P}_r(\alpha, m) = \sum_{h=b}^{\alpha b} \binom{\alpha b}{h} (1 - \frac{m}{N})^h \left(\frac{m}{N}\right)^{\alpha b - h}.
$$

(4.9)

Recall that $\alpha b$ in Eq (4.9) has to be an integer. Similar to SolarCode, we can allow $\alpha$ to be any real number in $[1, +\infty)$, by asking the coding module to always generate $\lfloor \alpha b \rfloor$ encoded blocks, and generate one extra encoded block with probability $\alpha b - \lfloor \alpha b \rfloor$. Thus, the successful recovery probability function for a general $\alpha$ is

$$
Pr(\alpha, m) = (1 + \lfloor \alpha b \rfloor - \alpha b)\widetilde{P}_r\left(\frac{\lfloor \alpha b \rfloor}{b}, m\right) + (\alpha b - \lfloor \alpha b \rfloor)\widetilde{P}_r\left(\frac{\lfloor \alpha b \rfloor}{b}, m\right).
$$

(4.10)

Let $T$ be the time interval from the present to the time when the next uploading opportunity appears. We discretize $T$ into $N_t$ slots and each slot has a length of $\Delta = T/N_t$. The probability that the next failure happens in the $r^{th}$ slot is

$$
P_1(0)^{r-1} \times (1 - P_1(0)),
$$

which means that all $N$ nodes are alive with probability $P_1(m = 0)$ for consecutive $r - 1$ time slots and then at least one out of the $N$ nodes fails in the $r^{th}$ slot. Note that $r$ could be smaller than $N_t$. When that happens, we will then re-evaluate the problem right away with a new $T$, which is from that time to the next uploading opportunity. On the other hand, $r$ could also be larger than $N_t$. But we again only consider a time duration of $T$ for our problem, and will re-evaluate the problem again after $T$. Therefore, the length of each evaluation cycle follows a geometric distribution with a cut-off threshold $N_t$. The probability
that a cycle has \( r \) slots can then be expressed as:

\[
P_2(r) = \begin{cases} 
P_1(0)^{r-1} \times (1 - P_1(0)), & \text{if } 1 \leq r < N_t; \\
P_1(0)^{r-1}, & \text{if } r = N_t. 
\end{cases}
\] (4.11)

Hence, the evaluation cycle of the problem is the time interval from now to the next node failure or to the next upload opportunity, whichever comes earlier.

Our objective is to maximize the total expected utility of the recoverable data that we can obtain during an evaluation cycle. With a probability of \( P_2(r) \), the evaluation cycle has \( r \) time slots. Moreover, with a probability of \( P_1(m) \), \( m (0 \leq m \leq N) \) nodes fail, and the collected data can be recovered with a probability of \( Pr(\alpha, m) \). The replication factor \( \alpha \) should be adaptively adjusted according to the node energy/storage status and the utility category. Thus we use \( \alpha_i^k(t) \) to denote the replication factor of the data in utility category \( k \) on node \( i \) at time slot \( t \). Therefore, formally, our overall objective can be expressed as:

\[
\max \sum_{r=1}^{N_t} P_2(r) \sum_{m=0}^{N} \left[ P_1(m) \sum_{t=1}^{r} \sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}} U^k R_i^k P_r(\alpha_i^k(t), m) \Delta \right],
\] (4.12)

subject to the following constraints.

First, similar as in the formulation of SolarCode in Chapter 3, the residual storage and residual energy of node \( i \) at time slot \( t \), denoted as \( s_i(t) \) and \( e_i(t) \) respectively, should satisfy a non-depletion constraint:

\[
s_i(t) > 0, \quad e_i(t) > 0, \quad \forall i \in \mathcal{N}, 1 \leq t \leq N_t.
\] (4.13)

In order to have feasible solutions, the system needs to have no storage or energy depletion originally when data replication is not used. This can be enforced in the design of the actual system. In other words, we assume that the non-depletion constraint is satisfied when all \( \alpha_i^k(t) \)'s are equal to one.
Second, the residual storage of node $i$ at time slot $t$ equals to the residual storage at the last time slot $t - 1$ minus the amount of storage consumed within time slot $t$. The total amount of encoded data generated on all $N$ nodes during time slot $t$ is $\sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}} R^k_i \alpha^k_i (t) \Delta$. As all encoded blocks are evenly distributed among the $N$ nodes, we have this storage evolution constraint:

$$s_i(t) = s_i(t - 1) - \frac{1}{N} \sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}} R^k_i \alpha^k_i (t) \Delta, \forall i \in \mathcal{N}, 1 \leq t \leq N_t. \quad (4.14)$$

Although this constraint is expressed in the manner of expectation, according to the law of large numbers, it holds with a high probability as the number of encoded blocks generated is large.

Third, the residual energy of node $i$ at time slot $t$ equals to the remaining energy at the last time slot $(t - 1)$ plus the solar energy harvested less the consumed energy within time slot $t$. The two major components of energy consumption in sensor networks are by CPU and wireless radio. Let $C_i$ be the CPU power consumed by data collection processes together with system processes running on node $i$. Since erasure codes use the exclusive OR operation ($\oplus$) to encode the data, they are very computationally efficient. Thus, we ignore the extra CPU load brought about by the coding process.

The energy consumed by the wireless radio depends on the data transmitted, received and overheard. During each time slot $t$, for each of its encoded blocks, node $i$ shifts it to a node $j$ that is randomly chosen from $\mathcal{N}$. We assume that the routes of data shifting are determined by some other routing module and is not considered as an optimization knob in this work. Let $u_{ij}$ be the route used to forward the data from $i$ to $j$, and $\mathcal{U} = \{ u_{ij} : i, j \in \mathcal{N} \}$ be the set of all routes used by all the node in the network. Since node $i$ evenly distributes its encoded blocks among the $N$ nodes, the expected data rate of route $u_{ij}$ at time slot $t$ is $d_i = \frac{1}{N} \sum_{k \in \mathcal{K}} R^k_i \alpha^k_i (t)$. Let $\mathcal{L}$ be the set of all links in the network, $l_{xy} \in \mathcal{L}$ be a directional link from node $x$ to node $y$, and $a_{xy}$ be the successful transmission probability of link $l_{xy}$. We write $l_{xy} \in u_{ij}$ to denote that link $l_{xy}$ is on route $u_{ij}$. Thus, the expected incoming traffic
rate of node $x$ is

$$
I_x = \sum_{l_{yx} \in \mathcal{L}} \sum_{u_{ij}, l_{yx} \in u_{ij}} \frac{d_i}{a_{yx}},
$$

which is the summation of the traffic on each incoming link $l_{yx}$ of node $x$, and where $d_i$ is the data rate of flow $u_{ij}$ that passes through link $l_{yx}$, and the factor $1/a_{yx}$ accounts for the data retransmission due to collisions. Similarly, the outgoing traffic rate of node $x$ is

$$
O_x = \sum_{l_{xy} \in \mathcal{L}} \sum_{u_{ij}, l_{xy} \in u_{ij}} \frac{d_i}{a_{xy}}.
$$

Moreover, because of the broadcast nature of wireless transmission, a node could overheard transmissions that are not intended for itself. Let $\mathcal{V}_x$ be the set of nodes whose transmission can be overheard by node $x$. Thus, the expected traffic rate overheard by node $x$ is

$$
H_x = \sum_{z \in \mathcal{V}_x} \sum_{l_{zy} \in \mathcal{L} - \{l_{zx}\}} \sum_{u_{ij}, l_{zy} \in u_{ij}} \frac{d_i}{a_{zy}}.
$$

Hence, the energy consumption rate of node $i$ is

$$
W_i(t) = C_i + P^tx_i O_i + P^{rx}_i (I_i + H_i),
$$

where $P^tx_i$ and $P^{rx}_i$ is the power consumption rate of the wireless radio for data transmitting and receiving, respectively. Again, according to the law of large numbers, Eq (4.15) holds with a high probability.

Similar as in Chapter 3, we assume that the available energy for recharging the battery of node $i$ at time $t$, denoted as $S_i(t)$, is known as an input of the problem. Recall that $S_i(t)$ may not be fully harvested into the battery because of the battery capacity bound. Namely, $e_i(t)$ should also be bounded by the battery capacity $B_i$. Thus, we have an energy evolution constraint:

$$
e_i(t) = \min\{e_i(t - 1) + (S_i(t) - W_i(t)) \Delta, B_i\}, \forall i \in \mathcal{N}, 1 \leq t \leq N_t.
$$
The more energy is in the battery, the higher is the reliability that can be achieved, which means that the objective function is non-decreasing with $e_i(t)$. Therefore, the energy evolution constraint is equivalent to the following two linear constraints:

\[
e_i(t) \leq e_i(t-1) + (S_i(t) - W_i(t))\Delta,
\]
\[
e_i(t) \leq B_i, \quad \forall i \in \mathcal{N}, 1 \leq t \leq N_t.
\]

In addition to the above constraints, all replication factors $\alpha_i^k(t)$ have to be greater than or equal to one in order to produce meaningful erasure codes:

\[
\alpha_i^k(t) \geq 1, \quad \forall k \in \mathcal{K}, i \in \mathcal{N}, 1 \leq t \leq N_t.
\]

Lastly, the initial storage and energy values $s_i(0)$ and $e_i(0)$ ($i \in \mathcal{N}$) are given as inputs.

Note that we do not have to count the term when $m = 0$ (i.e., no node failure) in the objective function Eq (4.12), because all the collected data are recoverable in this case and thus it is just a constant term with no effect on the optimization problem. Moreover, we assume in Eq (4.12) that multiple node failures could happen within each time slot. In practice, however, $P_1(m)$ for $m > 1$ is usually very small so that the cases of multiple node failures can be neglected because of their small occurrence probabilities. Therefore, for computational efficiency, we can focus only on the case with a single node failure ($m = 1$), and simplify the objective function as

\[
\max_{r=1}^{N_t} \sum_{r} P_2(r) \sum_{t=1}^{r} \sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}} U_i^k R_i^k P_r(\alpha_i^k(t), 1)\Delta.
\]

Recall that the length of an evaluation cycle may be shorter than $T$, while the constraints of the formulated problem apply to the whole time duration $T$. This means that we might have a better solution if we knew when exactly a node failure would happen and then removed the constraints after the failure time for that node. Consequently, this node could allocate
all its energy and storage resources for its operation before the failure and thus achieve better performance. However, it is impossible for us to predict node failures. Actually, the probability that the evaluation cycle is shorter than $T$ will be very small as the failure probability is small. Therefore, the current formulation gives us close-to-optimal solutions.

4.3.2 Solution

Based on Theorem 2 in Chapter 3, it is straightforward to prove the following theorem.

**Theorem 3.** $Pr(\alpha, m)$ is always concave with respect to $\alpha$ if the number of sensor nodes $N$ satisfies

$$
N \geq \frac{(b + 1)(b + 2)}{3(b + 1) - \sqrt{3(b^2 - 1)}} \cdot m. 
$$

(4.19)

Although the lower bound in Theorem 3 is not tight, it is below 17 even for a very large $b = 19$ and a typical $m = 1$, as shown in Figure 4.9. Considering that $b$ is usually not too large because of the extra framing overhead introduced by encoded blocks, we can see that this lower bound on the network size can be easily satisfied. Even if $N$ is lower than the bound, we can adjust $b$ accordingly to still ensure the concavity of the objective function.
Based on Theorem 3, we propose SolarQoS to determine the redundancy levels of erasure codes. Similar as SolarCode, it first computes the line functions $\ell(i)$ used in Eq (3.13). Plugging $Pr(\alpha, m)$ into the objective function, we obtain a normal convex optimization problem, and solve it for $\alpha_i^k(t), i \in \mathcal{N}, k \in \mathcal{K}, 1 \leq t \leq N_t$.

SolarQoS is called every time when major events happen in the network, including stored data are collected or some node fails. It runs in a centralized manner on the node with the most remaining energy in its battery, and the resulting $\alpha_i^k(t)$ are sent to the corresponding nodes. The complexity for solving the optimization problem basically depends on its number of variables and constraints. For SolarQoS, it has $NN_t|\mathcal{K}| \alpha_i^k(t)$s, $NN_t s_i(t)$s and $NN_t e_i(t)$s as variables, and $5NN_t + NN_t|\mathcal{K}|$ constraints. For a typical problem with $N = 9$, $|\mathcal{K}| = 4$ and $N_t = 168$, SolarQoS takes an average of 51 seconds when running on a personal computer with a 2.4GHz CPU. Although it may take a longer time when running on an embedded PC (our solar-powered nodes use PC-class processors), the speed of solving for the replication factors is not critical for long-running applications, since the computation needs to be performed only very infrequently (e.g., of the order of changes in weather forecasts), which makes the execution time of SolarQoS acceptable.

### 4.3.3 Performance Evaluation

We evaluate its performance based on the real settings of an existing solar-powered sensing testbed. As most deployments in practice, this testbed has a small scale and nodes are all within the communication range of each other.

In our performance evaluation, we consider the following four applications: (1) **AUDIO** collects bird vocalizations, (2) **VIDEO** monitors motion in bird nests, (3) **SOLAR** records the output current and voltage of the solar panel, and (4) **TEMP** logs the temperature inside the node enclosure. **SOLAR** and **TEMP** have the same sampling rate of 1Hz and keep all the samples, while **AUDIO** and **VIDEO** sample at rates of 11KHz and 2Hz respectively, but only keep the data that are potentially interesting to the end users (e.g., audio clips
with bird vocalizations, or video clips with motions). Therefore, the data collection rate may be different from the sampling rate as noise data are discarded. Specifically, SOLAR (TEMP) has the same data collection rates on all nodes, while the data collection rate of AUDIO (VIDEO) on different nodes varies, depending on the node locations. Based on the past experiments of AUDIO and VIDEO on the testbed, we summarize their average data collection rates in Table 4.2, which also shows the rates of SOLAR and TEMP.

Table 4.2: Data collection rates (byte/s) of the four applications on the 9 nodes.

<table>
<thead>
<tr>
<th></th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
<th>$n_4$</th>
<th>$n_5$</th>
<th>$n_6$</th>
<th>$n_7$</th>
<th>$n_8$</th>
<th>$n_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIO</td>
<td>10.3</td>
<td>8.1</td>
<td>9.9</td>
<td>7.5</td>
<td>5.3</td>
<td>8.7</td>
<td>6.2</td>
<td>11.8</td>
<td>6.4</td>
</tr>
<tr>
<td>SOLAR</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>VIDEO</td>
<td>14.2</td>
<td>9.8</td>
<td>10.2</td>
<td>10.4</td>
<td>12.2</td>
<td>14.1</td>
<td>11.9</td>
<td>14.1</td>
<td>10.2</td>
</tr>
<tr>
<td>TEMP</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The data collected by AUDIO are being studied by environmental scientists, and thus we assign AUDIO the highest utility. The solar energy harvesting traces created by SOLAR are utilized to facilitate the research on solar energy management, and we choose a lower utility level for it. The usage of VIDEO and TEMP is still in the early stages, so we assign them the lowest utility. Obviously, the behavior of SolarQoS depends on the absolute utility values assigned to each utility level. In the following experiments, we first (somewhat arbitrarily) set $U_{AUDIO} = 10$, $U_{SOLAR} = 5$ and $U_{VIDEO} = U_{TEMP} = 1$, and then study the effect of different utility values on SolarQoS in the end.

While our problem formulation can handle most failure models in general, in the experiments, we assume that node failures happen independently with a small probability $p$. Thus, we have $P_1(m) = \binom{N}{m} p^m (1 - p)^{N-m}$. The hourly node failure probability $p$ in our experiments is set as 0.0006, which implies that the expected time to the next failure is about 70 days. This is consistent with actual observations on the deployed testbed. Since failures occur very rarely on the testbed, we manually emulate them for the evaluation purpose by discarding the encoded data blocks on failed nodes. We choose $b = 8$ for the LT coding.
module. According to Theorem 3, this can ensure the concavity of the objective function Eq (4.12) when $N = 9$ for the testbed.

We first study how SolarQoS adjusts the replication factors of the four applications according to their utility levels and the environmental changes. In Figure 4.10, the top and middle part of the figure show the replication factors scheduled by SolarQoS on node 8 and node 9, which have the highest and lowest data collection rates, respectively. The bottom part of the figure shows the predicted solar energy trace for reader’s reference. Three interesting observations are in order.

First, the replication factors $\alpha$ of all four applications on both nodes are adaptively adjusted according to the energy level of the sensor nodes throughout the 7 days. As we can see, the $\alpha$s stay at a low level to save energy for avoiding blackout, when less solar energy is available in bad weather conditions (e.g., rain in day 4). On the other hand, the $\alpha$s are
Figure 4.11: The residual energy in the batteries of node 8 for the two cases when SolarQoS is used and not used.

Increased to higher levels during days of good weather. This is because there is an incentive to spend energy to replicate data when the battery is almost fully charged and extra solar energy is still available. Otherwise this energy surplus would be wasted (because the battery cannot store surplus energy when fully charged).

Second, let us compare the replication factors of the four applications on either node. As shown in Figure 4.10, the $\alpha$s of VIDEO and TEMP are always nearly 1 (i.e., no replication), except for the time when the residual energy is high. Meanwhile, AUDIO always has the highest $\alpha$ among all the applications. When solar energy becomes available, AUDIO is the first to increase its $\alpha$. When saving energy is necessary during nights and days of bad weather, SOLAR lowers its $\alpha$ earlier than AUDIO. Also, VIDEO and TEMP have almost the same $\alpha$, which means that the replication factor depends more on the utility value of an application rather than its data collection rate.

Third, we compare the replication factors between the two nodes. Note that node 8 and node 9 have the highest and lowest total raw data collection rates, respectively. Because of the heavy data load, each application on node 8 in general has a lower or equal replication factor $\alpha$ than the same application on node 9. Due to the same reason, applications on node 9 have more stable replication factors than those on node 8.

In order to study the effect of SolarQoS on the residual energy in the node battery, we plot the residual energy of node 8 in Figure 4.11 for the two cases when SolarQoS is used and not used. The residual energy of other nodes shows a similar characteristics, which are therefore
Table 4.3: Average replication factors ($\alpha$) of the four applications by SolarQoS with the predicted or real solar energy trace. The last column is the overall utility achieved.

<table>
<thead>
<tr>
<th></th>
<th>AUDIO</th>
<th>SOLAR</th>
<th>VIDEO</th>
<th>TEMP</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted trace</td>
<td>1.2626</td>
<td>1.1990</td>
<td>1.0227</td>
<td>1.0227</td>
<td>76251</td>
</tr>
<tr>
<td>Real trace</td>
<td>1.2719</td>
<td>1.1960</td>
<td>1.0170</td>
<td>1.0170</td>
<td>77977</td>
</tr>
</tbody>
</table>

omitted here for the limited space. Clearly, SolarQoS consumes extra energy and thus results in a lower residual energy than when it is not used, but the battery can still be fully charged at noons. This means that the extra energy cost by SolarQoS is just the energy surplus, which would be wasted (if not used) because of the limit of the battery capacity. Hence, SolarQoS does not affect the normal operation of the sensor network. Another interesting finding is that the residual energy reaches 100% in some days (e.g., day 1 and day 2) and stays at this full level for a short period of time, which implies that not all available energy surplus has been used by SolarQoS. The reason behind this is that the replication factor is not only bounded by the energy constraint, but also the storage constraint. This is the case when the storage constraint takes effect.

If the exact solar energy trace were known to an oracle, we could expect that SolarQoS could perform perfectly in allocating energy for data replication. We run the experiment again with the real solar energy trace. In Table 4.3, we compare the average replication factors, which are averaged over all nodes weighted by their data collection rates, and the total utility in the two cases. We can see that Solar with the predicted trace has comparable performance to that with the real trace. In fact, with the real trace, only AUDIO has a little higher average $\alpha$, while other three applications have slightly lower average $\alpha$s than those with the projected trace. This is due to the non-deterministic relation between the two traces. However, the overall utility obtained when using the real solar energy trace is a little bit higher. This is because SolarQoS reserves $3\sigma$ energy to deal with errors in solar energy prediction. Now with the exact energy trace as the input, SolarQoS could allocate more energy to enhance storage reliability and hence attain a higher total utility.
Finally, we compare the performance of SolarQoS with the scheme of no replication, and an adaptive scheme without considering the difference of the data utility. We call these two schemes NoRep and NoQoS, respectively. Figure 4.12 compares their data recovery ratios of each application, which equal to the amount of data that are recoverable from node failures over the amount of data collected by each application, and their overall utility recovery ratios, which are calculated as the utility of all the recoverable data over the utility of all the data collected. Recall that the overall utility is the expected utility of the recoverable data summing over possible cases, as calculated in Eq (4.12). For a fair comparison, even though NoRep does not replicate data, we allow it to still distribute the collected data evenly among all nodes so as to avoid storage depletion on some particular nodes because of the uneven data collection rates. Thus, NoRep behaves like SolarQoS with $\alpha = 1$ and equal utility values. As shown in Figure 4.12, the data of each application under NoRep have the same recovery ratio 39%, which equals to $Pr(\alpha = 1)$. Compared to NoRep, NoQoS uses data replication and thus achieves higher recovery ratios. However, NoQoS still treats data of different utility levels equally when making replication, so the applications under NoRep have the same

<table>
<thead>
<tr>
<th>[6,3,1,1]</th>
<th>[10,5,1,1]</th>
<th>[14,7,1,1]</th>
<th>[20,10,1,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SolarQoS/NoRep</td>
<td>147.23%</td>
<td>148.94%</td>
<td>149.84%</td>
</tr>
<tr>
<td>SolarQoS/NoQoS</td>
<td>108.32%</td>
<td>111.27%</td>
<td>112.82%</td>
</tr>
</tbody>
</table>

Table 4.4: The utility ratio of SolarQoS over NoRep and NoQoS under different utility assignments.
recovery ratios. In contrast, SolarQoS allocates more energy and storage resources for the data of high utility levels. As a result, AUDIO and SOLAR gain higher recovery ratios, while VIDEO and TEMP earn lower recovery ratios because of their low utility values. In terms of the overall utility recovery ratio, as shown by the experiment results, SolarQoS achieves 11.27% and 48.94% higher than NoQoS and NoRep, respectively. In fact, the utility of the recoverable data achieved by SolarQoS over the utility by NoRep and NoQoS depends on the absolute utility values of the applications. In Table 4.4, we summarize the utility gain of SolarQoS, where the numbers in each pair of the square brackets are the utility values of AUDIO, SOLAR, VIDEO and TEMP, respectively. It can be seen that the utility gains of SolarQoS increase when the distribution of the utility value becomes more skewed.

4.4 SimStore: A Content-aware Storage Service based on Dynamic Similarity

Although nodes on high-end sensing systems usually have reasonably large storage space, the storage could still be run out due to the high-bandwidth of the data collection process. For example, on our testbed, each node has a local disk of 16 GB. When collecting bird vocalizations in CD-quality (i.e., 44.1 KHz and 16 bits per sample), a node could run out of its disk in 50.39 hours if no data uploading opportunity appears during this period. Therefore, in this section, we focus on the problem which data should be discarded as new data are kept being collected and the storage has already been run out.

One traditional solution for this problem is the FIFO policy, which always replaces the oldest data with the newly collected one. This solution treats all data equally, ignoring the fact that not all data have the same criticality/utility to end users. For example, sensory data from different sensors or sensory data in different value ranges from the same sensor may have different utilities. To incorporate multi-utility in data storage, a popular method is to assign different utility values to data collected by different sensors or data with different
features. Then, the objective is to maximize the total utility of all the stored data. We have already explored this static utility model in Section 4.3.

Moreover, instead of being static, data utility may dynamically depend on the data collected in the past and by neighboring nodes. In another word, in some cases, assigning a static value as utility is not appropriate because utility could be dynamic in both temporal and spatial dimensions. For example, if similar data are kept being collected on one node or on some neighboring nodes over the past, the data become less interesting. More specifically, if multiple nodes detect a same object of interest and take pictures of the object from slightly different angles, then we argue that keeping all the pictures has very little utility gain than keeping just one or some of the pictures.

Therefore, our objective is to identify the most representative data and then keep the top \( k \) ones given \( k \) as the storage constraint. This problem is close to the clustering analysis in data mining area, which summarizes data such that similar objects are grouped together while dissimilar ones are separated. In particular, we can use a clustering algorithm to group the sensory data into \( k \) clusters, and then only keep the centers of the \( k \) clusters in the storage.

However, we can not directly apply traditional offline clustering algorithms because of the following facts. First, sensory data are collected incrementally as a time series, and hence not all data are known at the very beginning. Therefore, we need an algorithm that makes clustering decisions before all the data are available. During this incremental clustering process, it is challenging to guarantee the clustering performance at every step for all the data that have ever been collected. Second, clustering decisions are irrevocable. Namely, data that have been discarded because they were not cluster centers can no longer become cluster centers again later in the clustering process, even though they may become better center candidates after more data have been collected. Third, sensory data are collected distributively on multiple sensor nodes, and data collected on different nodes may have correlations when the nodes are detecting the same event. Thus, we need to coordinate the
clustering processes on distributed nodes and maintain a good overall clustering on these
distributed data that are continuously evolving.

In this section, we first formulate the problem. Then, we introduce an incremental clus-
tering based algorithm that each node can run to manage its local storage. Next, we extend
this algorithm to a distributed protocol, called SimStore, that coordinates the storage man-
age of multiple sensor nodes. Finally, we evaluate SimStore in the context of a bird
vocalization collection application.

4.4.1 Problem Formulation

We consider a sensing system with a node set $\mathcal{N}$, and write $N = |\mathcal{N}|$ to denote the system
size. Sensory data are collected on nodes and then stored in the network until the next
upload opportunity appears. We assume that each data sample has a fixed size $\Theta$. Given a
local storage of size $\text{Storage}_i$, node $i \in \mathcal{N}$ can store at most $k_i = \lfloor \frac{\text{Storage}_i}{\Theta} \rfloor$ data samples in
its local storage.

Without losing generality, we assume that every node runs one same application, which
has a fixed sampling rate of 1. Let $\mathcal{X}_i = \{x_i(0), x_i(1), \ldots, x_i(t), \ldots\}$ be the infinite time-series
of the data samples collected on node $i \in \mathcal{N}$, where $x_i(t)$ is the data sample collected at time
$t$ on node $i$. We extract multiple features for each data sample $x_i(t)$, and then each $x_i(t)$ can
be viewed as a data point in a multi-dimensional space based on its features. We denote the
distance between two multi-dimensional data points $x$ and $y$ in this feature space as $d(x, y)$.
Our proposed scheme is, in general, independent of the features used to represent the data
samples and the definition of distance. Therefore, we delay their definition in Section 4.4.4
when we evaluate SimStore with a bird vocalization collection application. Comparing to
the massive size of raw data samples (e.g., video or audio clips), the size of a feature value is
typically only few bytes, and thus we ignore the storage space needed to store the features
of the data samples on each node.

In this work we concentrate on the $k$-centering objective for clustering, since this is a
simple and popular criterion for clustering in arbitrary metric spaces. Traditional offline
$k$-centering assumes that all the data inputs are available at the beginning. For example,
given a finite set $X$ of data samples, we can define the $k$-centering problem as follows.

**Definition 1.** The $k$-centering problem is, given a set $X$ of $t$ points and an integer $k \leq t$,
to identify a set $C \subseteq X$ of $k$ points as centers such that $\max_{x \in X} \min_{c \in C} d(x, c)$ is minimized,
which means that the maximum of the distance from a point $x \in X$ to $C$, defined as $d(x, C) = \min_{c \in C} d(x, c)$, is minimized.

Given the cluster centers, a data point $x$ belongs to the cluster with center $c$ (denoted as $x \in O_c$), to which it has the shortest distance $d(x, c)$. We define the radius of a cluster as the maximum distance from the center to any point in the cluster. Then, the $k$-centering problem is essentially to minimize the maximum radius of all the $k$ clusters. We define the maximum radius of the $k$ clusters as the performance (or cost) of a $k$-centering result.

Since sensory data are collected incrementally and form an infinite time series $X_i$, we need to make clustering decisions at every time when a new data sample $x_i(t)$ is collected. We define this incremental $k$-centering problem as following

**Definition 2.** Given an infinite time series $X = \{x(0), x(1), ..., x(t), ...\}$, the incremental $k$-centering is to maintain a collection of $k$ clusters such that as each data point $x(t)$ is collected, either it is assigned to one of the current $k$ clusters, or it triggers a reorganization of the existing clusters.

We define the performance of an incremental $k$-centering algorithm as the maximum ratio, over any time $t$, of the cost of the $k$-centering result to the cost of the optimal $k$-centering for the input $X = \{x(0), x(1), ..., x(t)\}$. This definition enforces that the cost of the incremental clustering result at any time instance $t$ needs to be considered for the sake of the overall performance.

In this work, we assume that all the nodes in $\mathcal{N}$ are within the communication range of each other. In case of large scale systems where nodes are multiple hops from each other,
we can group the nodes into different groups, and employ the propose scheme within each group. This grouping scheme can be justified by the fact that nodes that are far away from each other are very unlikely to detect the same event and thus their data have no strong correlations.

4.4.2 Local Storage Replacement Policy

We first consider a simple scenario where each node manages its local storage without coordinating with other nodes. Many incremental $k$-centering algorithms have been proposed in the literature, we incorporate one called Doubling Algorithm (DA) [60] proposed for clustering streaming data because it only requires a single pass of the data and directly forgets the input data and only maintains the cluster centers. Based on DA, we propose a local storage replacement policy called Local-DA as outlined in Algorithm 3.

Let $C$ be the set of the current cluster centers. Before we have collected $k$ data samples, we always have storage space to store the samples, and let $D$ be initialized as the smallest pairwise distance between them. Afterwards, given a new data sample $x$, we examine whether $x$ is close enough to the existing $k$ cluster centers by using a radius of $2D$. If $d(x, C) \leq 2D$, we simply discard $x$ because it is not a representative data point. If $d(x, C) > 2D$, we need to store this new sample. If the current size of $C$ is less than $k$, then there is still space available for storing $x$, and hence we store $x$ and augment $C$ with this new center $x$. Otherwise, we need to reorganize the $k+1$ clusters $C \cup \{x\}$. Specifically, we arbitrarily pick one center $c$ and then discard all other centers that are close to (within $2D$ of) $c$. We repeat this process until all centers are separated by at least $2D$. Then, we double $D$ for the next round.

As proved in [60], Local-DA guarantees an 8-factor performance ratio to the optimal clustering at any given point in the data stream. Intuitively, although the clustering radius is doubled after every cluster reorganization, it can always be bounded within a range of the optimal radius. Specifically, before the center reorganization, all data points are within
Algorithm 3 Local-DA \( (k, x, t, C, D) \)

Input: the local storage space \( k \), the current data sample \( x \) and its index \( t \), the set of the current cluster centers \( C \), and center distance \( D \);

Output: the updated \( C \) and \( D \);

1: if \( t \leq k \) then
2: Store \( x \) on the local disk;
3: \( C = C \cup \{ x \} \);
4: \( D = \) the smallest distance between the centers in \( C \);
5: return \( (C, D) \);
6: end if
7: if \( d(x, C) \leq 2D \) then
8: Discard \( x \);
9: return \( (C, D) \);
10: end if
11: if \( |C| < k \) then
12: Store \( x \) on the local disk;
13: \( C = C \cup \{ x \} \);
14: return \( (C, D) \);
15: end if
16: \( C' = \{ \} \);
17: for \( c \in C \cup \{ x \} \) do
18: if \( d(c, C') \leq 2D \) then
19: Delete \( c \) from the disk;
20: else
21: \( C' = C' \cup \{ c \} \);
22: end if
23: end for
24: return \( (C', 2D) \);

\( \sqrt[3]{2D} \) of the current centers. When a data is outside \( 2D \) of the current centers, we reorganize the centers such that centers within \( 2D \) are merged to one new center. Thus, by applying triangulation inequality, all the data points that have ever been collected so far are within \( 2D + 2D = 4D \) of the new centers. This implies that the cost of the clustering by Local-DA is at most \( 4D \). Also, when a new data point \( x \) can not be covered by the existing clusters, the distance between these \( k + 1 \) data points are at least \( 2D \), which means that the cost of the optimal clustering \( R_{OPT} \) is at least \( D/2 \). Therefore, the cost of the clustering by Local-DA is not greater than \( 8R_{OPT} \).
4.4.3 Distributed Storage Replacement Protocol

Distributed clustering algorithms [61, 62] have been proposed in prior work, but mainly focus on the case where the global clustering renders the same clustering size as the local clustering does. Namely, each node $i$ runs a local algorithm and reports the obtained $k$ centers to a central coordinator, which runs another clustering algorithm on the reported $kN$ local centers to obtain the final $k$ centers. It has been proved in [61] that the global clustering has an $a_1 + a_2$ approximation ratio if the local clustering gives an $a_1$-approximation and the one on the coordinator gives an $a_2$-approximation.

In our problem, however, the overall storage of the whole sensing system is simply the union of the local storage of all the nodes. Therefore, the size of the global clustering is the sum of ones of all the local clusterings. Namely, in our problem, on the coordinator site, it is looking for $\sum_{i \in N} k_i$ cluster centers from the $\sum_{i \in N} k_i$ data points reported from all the nodes. Therefore, no further clustering is needed, and the global clustering result is simply the union of the results of all the nodes. Unfortunately, in this case, we no longer have a finite performance bound on the global clustering. This can be easily illustrated by the example in Figure 4.13. Suppose that both node 1 and node 2 cluster their two local data points into $k = 1$ cluster. In a global view, our goal is to cluster all the four data points into $k = 2$ clusters. As we can see in Figure 4.13, if the data points on one node are very close to (or even overlap with) those on the other node, the cost of the optimal global clustering could be arbitrarily smaller than the cost of the simple union of the local clusterings.

Therefore, for the sake of overall performance, instead of using the above approach with two separated steps, individual nodes have to coordinate with each other throughout the clustering process. In this section, we propose a protocol called SimStore, which enables distributed nodes to run the Doubling Algorithm in a globally coordinated fashion. The main design objective of SimStore is then to minimize the communication overhead incurred in the node coordination. The basic idea is that each node maintains its local clustering
result, while one node is dynamically selected as the coordinator to maintain a global view of the clustering. When a new data sample is collected on a node, unless a cluster reorganization is required, a cluster decision can be made locally without affecting other nodes. When a cluster reorganizing is needed, it is performed on the coordinator node and then the result is passed to each individual node so that the obsoleted centers can be removed from the local disk. Next, we will describe the protocol on the node side and the coordinator side respectively.

A. On the Node Side
Algorithm 4 SimStore-Node($k_i, x_i, t_i, C_i, D_i$)

**Input:** the local storage space $k_i$, the current data sample $x_i$ collected by node $i$ and its index $t_i$, the set of the current local cluster centers $C_i$, and the cluster distance $D_i$;

**Output:** the updated $C_i$ and $D_i$;

1: if $d(x_i, C_i) \leq 2D_i$ then
2: Discard raw $x_i$;
3: return $(C_i, D_i)$;
4: end if
5: $ACK = Report$(Candidate, $x_i$); //report a new cluster candidate to coordinator
6: if $ACK ==$ discard then
7: Discard $x_i$;
8: else if $ACK ==$ keep then
9: Store raw $x_i$ on the local disk;
10: $C_i = C_i \cup \{x_i\}$;
11: else if $ACK ==$ shift to $j$ then
12: Send raw $x_i$ to node $j$;
13: else if $ACK ==$ reorganized then
14: $(C'_i, D_i) =$ the new cluster centers and radius from coordinator.
15: for $c \in C_i \cup \{x_i\}$ do
16: if not $c \in C'_i$ then
17: Delete raw $c$ from the disk;
18: end if
19: end for
20: $C_i = C'_i$;
21: end if
22: return $(C_i, D_i)$;

Figure 4.14 shows an overview of SimStore. Each node $i$ periodically collects sensory data sample $x_i$ \(^2\) and then calls SimStore-Node as described in algorithm 4 to decide whether to keep or discard shift this sample, or shift it to other nodes. Besides storing the raw data samples, each node $i$ also stores the extracted features of the stored raw data, which are the centers of the current local clusters. Let $C_i$ be the set of local cluster centers and $D_i$ be the shortest distance between the first $k$ data samples.

Given a new data sample $x$, if $x_i$ is within $2D_i$ of the local centers, then we immediately know that we can discard $x_i$ without hurting the global clustering. If $x_i$ can not be covered by local centers, then we have to consult the coordinator. If the coordinator replies a decision

\(^2\)We use $x_i$ to denote the extracted features of the raw data sample, and use “raw $x_i$” to denote the raw data sample.
of *discard*, it means that this new sample can be covered by a cluster on some other node and hence we can discard it. If the coordinator replies a decision of *keep*, it means that this new sample is representative and hence node $i$ can keep it\(^3\). If the coordinator replies a decision of *shifting* to $j$, it means that $x_i$ is a representative sample and node $j$ still has storage space for it. If the coordinator replies that a cluster reorganization has happened, then node $i$ uses the updated cluster centers $C'_i$ and distance $D_i$ to scan through its old cluster centers $C_i$. If an old center has been removed, then we remote the corresponding raw data from the local disk.

Note that shifting raw data from one node to another takes much more energy than exchanging data features between nodes and the coordinator. For example, in audio collection applications, a CD-quality audio clip of 30 seconds is 2.65 MB, while a 100-dimensional feature vector may only take a few hundred bytes. In order to save communication cost, we allow raw data shifting only when nodes have enough residual energy. We will elaborate on how the coordinator makes such decisions later in Section 4.4.3.B.

**B. On the Coordinator Side**

One node in $\mathcal{N}$ is dynamically selected as the coordinator based on its energy status. We will elaborate on how to select the coordinator later in Section 4.4.3.C. The coordinator maintains the local solutions $C_j$ for each node $j$ and the current global clustering radius $D$. As described in Algorithm 5, it reacts on receiving messages from other nodes in the network.

When the coordinator receives a report of a cluster center candidate $x_i$ from node $i$, we first check if $x_i$ can fit in one of the clusters on other nodes. If so, we can inform node $i$ to just discard $x_i$. Otherwise, it first checks if node $i$ still has space space left for this new center. If so, it inform node $i$ to keep $x$. Otherwise, if there are some other node $j$ that

\(^3\)As will be described in Section 4.4.3.B, the coordinator has already checked that node $i$ still has storage space left.
Algorithm 5 SimStore-Coordinator($k_j, C_j$ for $\forall j \in \mathcal{N}$)

**Input:** the local storage space $k_j$, the set of local cluster centers $C_j$ for each node $j \in \mathcal{N}$; 
Receive a report of a cluster candidate $x_i$ from node $i$

**Output:** the updated $C_j$ and $D$;

1. **if** $d(x_i, \cup_{j \in \mathcal{N}} C_j) < 2D$ **then**
2. \hspace{1em} ACK(discard, $x_i$);
3. \hspace{1em} **return** ;
4. **end if**
5. **if** $|C_i| < k_i$ **then**
6. \hspace{1em} ACK(keep, $x_i$);
7. \hspace{1em} **return** ;
8. **end if**
9. **if** $\exists j \in \mathcal{N} : |C_j| < k_j$ **then**
10. \hspace{1em} if Energy_Allow(i, j) **then**
11. \hspace{2em} ACK(shift, $x_i, j$);
12. \hspace{1em} **else**
13. \hspace{2em} ACK(discard, $x_i$);
14. \hspace{1em} **end if**
15. **else**
16. $C'_j = \{\}, \forall j \in \mathcal{N}$;
17. **for** $c \in \cup_{j \in \mathcal{N}} C_j \cup \{x\}$ **do**
18. \hspace{1em} **if** $d(c, \cup_{j \in \mathcal{N}} C'_j) \leq 2D$ **then**
19. \hspace{2em} $C'_j = C'_j \cup \{c\}$;
20. \hspace{1em} **end if**
21. **end for**
22. $C_j = C'_j, \forall j \in \mathcal{N}$;
23. $D = 2D$;
24. ACK(reorganized, $C_i, D$);
25. **end if**

still has storage space, it employs a threshold-based control policy (similar to SolarStore in Chapter 4.2) to decide if there is energy surplus on node $i$ and $j$ allows to shift $x_i$.

Specifically, let $P^{sys}_i$ and $P^{solar}_i$ be the average power consumption rate and power charging rate of node $i$, respectively. Let $B_i$ be the battery capacity of node $i$. Then, as in SolarStore, we can derive an energy surplus threshold $\frac{P^{sys}_i}{P^{solar}_i} B_i$, above which the residual energy can be regarded as energy surplus whose usage implies no impact on the node lifetime. Moreover, let $P^{tx}_i$ and $P^{rx}_i$ be the power consumption rate of the wireless radio for data transmitting and receiving on node $i$. So it costs $P^{tx}_i \Theta$ on node $i$ and $P^{rx}_j \Theta$ on node $j$ to shift a raw data
sample of size $\Theta$. Then we define the $\text{Energy}_{\text{Allow}}(i, j)$ to be true when the residual energy $e_i$ and $e_j$ of node $i$ and $j$ after the shifting are both above the energy surplus threshold; namely,

$$\text{Energy}_{\text{Allow}}(i, j) = \begin{cases} 
\text{true} & \text{if } e_i - P_{tx}^i \Theta \geq \frac{p_{sys}^i}{p_{total}^i} B_i \land e_j - P_{rx}^j \Theta \geq \frac{p_{sys}^j}{p_{total}^j} B_j \\
\text{false} & \text{otherwise}
\end{cases}$$

If the local disk of every node is full, we perform a cluster reorganization on all cluster centers as in Local-DA. Then, we inform the updated cluster centers $C_i$ and distance $D$ to node $i$.

C. Selecting the Coordinator

As described above, the coordinator keeps track of the residual energy $e_i$ of each node $i$. Since the coordinator works as a hub to receive and respond the node inquiries, SimStore-Coordinator consumes more energy than SimStore-Node. Thus, we select the node with the most residual energy as the coordinator. To avoid frequent switching of the coordinator, we change the coordinator only when its residual energy is below a range of that of another node. Specifically, let node $\textit{crd}$ be the current coordinator, if we find another node $i \in \mathcal{N} - \{\textit{crd}\}$ such that $e_i \geq e_j$ for $\forall j \in \mathcal{N} - \{\textit{crd}\}$ and $\nu e_i \geq e_{\textit{crd}}$, where $0 < \nu \leq 1$ is a ratio used for stability, then we switch the current coordinator to node $i$, and move the current clustering results $C_j$ for $\forall j \in \mathcal{N}$ and $D$ to the new coordinator $i$.

4.4.4 Performance Evaluation

We evaluate SimStore with a bird vocalization collection application. The collected data are used by environmental scientists to study the bird habitats. On each node, the application continues to sample a microphone in half CD-quality and then stores samples of every 30 seconds as a clip, whose size is 1.32 MB. Given a limited local storage, our objective is to
collect as many different bird vocalization events as possible. Different vocalization events may be made by different bird species or the same species but with some call variations. The multiple audio clips recorded for one event may have slightly different sound waves because of the different signal gains, background noises or even the natural variations in bird syllables over time. Therefore, we can not use exact waveform matching to find the similar audio clips belonging to the same event.

In order to know the ground truth, in the experiments, we select and use audio clips recorded for three bird species: red-winged blackbird, American crow, and song sparrow. They are among the most frequently observed species around the place where the testbed is deployed. Their appearance and some examples of their calls’ waveforms are shown in

Figure 4.15: The examples of appearance and call waveform of the three bird species.
<table>
<thead>
<tr>
<th>Bird Species</th>
<th>Number of Events</th>
<th>Number of Audio Clips</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red-winged Blackbird</td>
<td>12</td>
<td>366</td>
</tr>
<tr>
<td>American Crow</td>
<td>10</td>
<td>350</td>
</tr>
<tr>
<td>Song Sparrow</td>
<td>20</td>
<td>714</td>
</tr>
</tbody>
</table>

Table 4.5: The number of events and associated audio clips used in the experiments.

Figure 4.15. Table 4.5 summarizes the number of events that we labeled within each bird species, as well as the total number of associated audio clips that are used in the experiments. As the ground truth, we regard each of the total 42 events as one cluster, and regard audio clips associated with the same event belonging to the same cluster.

A. Feature Extraction

We adopt the most widely used feature extraction method for clustering/classifying acoustical data to extract a feature vector for each audio clip [63, 64].

Each audio clip is divided into \( M \) frames of equal length \( L \), with 50% overlaps in each of the two adjacent frames. We use \( M = 300 \), and \( L = 220500 \) (the number of samples in 100 ms). To improve the signal to noise ratio (SNR), the sample \( s_i \) \((1 \leq i \leq L)\) in each frame is scaled by a hamming window \(0.54 - 0.46 \cos(2\pi i/(L - 1))\). Then, for each frame, we compute its Discrete Fourier Transform (DFT) coefficients \( F(\omega) \), and extract the following features.

- **Total Power**: \( TP = \sum_{\omega=1}^{\omega_0} |F(\omega)|^2 \), where \( |F(\omega)|^2 \) is the power at the frequency \( \omega \) and \( \omega_0 \) is the half sampling frequency.

- **Subband Powers**: We divide the frequency spectrum into four subbands: \([0, \omega_0/8]\), \([\omega_0/8, \omega_0/4]\), \([\omega_0/4, \omega_0/2]\) and \([\omega_0/2, \omega_0]\), and then calculate the power within each subband, obtaining 4 features \( SP[1..4] \).

- **Spectrum Centroid**: \( SC = (\sum_{\omega=1}^{\omega_0} \omega |F(\omega)|^2)/TP \). It is also called brightness in the context of human perception. A brighter sound has a higher centroid.

- **Signal Bandwidth**: \( SB = \sqrt{\sum_{\omega=1}^{\omega_0} (\omega - SC)^2 |F(\omega)|^2}/TP \), which is the square root of
the power-weighted average of the squared difference between the spectral components and the spectrum centroid.

- *Mel-frequency Cepstral Coefficients (MFCC)* [65]: MFCCs are coefficients that collectively form a representation of the short-term power spectrum of a sound, and are commonly used as features in audio similarity measures. First, we use a set of $W$ triangular overlapping windows to map the powers of the spectrum (i.e., $|F(\omega)|^2$) onto the mel scale. Denoting the output of the $w^{th}$ window as $A_w$ ($w = 1, 2, ..., W$), we can calculate the MFCCs as $\text{MFCC}(m) = \sqrt{\frac{2}{W} \sum_{w=1}^{W} (\log A_w) \cos[m(w - 0.5)\pi/W]}$ for ($m = 1, 2, ..., F$), where $F$ is the total number of coefficients used. We set $D = 24$ in our experiments.

After extracting the feature $TP$, $SP[1..4]$, $SC$, $SB$ and $MFCC[1...F]$ for all the $M$ frames of an audio clip, we then calculate the means and standard deviations for each type of the features over all the $M$ frames, and use these $2(7 + F)$ values to form a feature vector for this audio clip. Figure 4.16 shows an example of how audio clips are mapped into a two-dimensional feature space formed by the mean $TP$ and mean $SC$. Note that the feature values are normalized by using z-scores, which we will elaborate in the next subsection.
B. Distance Function

Multiple distance functions are available for measuring the similarity/dissimilarity of two data points in a feature space. For example, there are Euclidean distance ($d_{Euc}$), Manhattan distance ($d_{Mht}$), and weighted Manhattan distance ($d_{wMht}$), which are defined as follows respectively.

\[
d_{Euc}(x, y) = \sqrt{\sum_{h=1}^{H} (x_h - y_h)^2}, \tag{4.20}
\]

\[
d_{Mht}(x, y) = \sum_{h=1}^{H} |x_h - y_h|, \tag{4.21}
\]

\[
d_{wMht}(x, y) = \sum_{h=1}^{H} \frac{|x_h - y_h|}{1 + |x_h| + |y_h|}, \tag{4.22}
\]

where, $x = (x_1, x_2, ..., x_H)$ and $y = (y_1, y_2, ..., y_H)$ are two $H$-dimensional data points.

Note that, when calculating distances between data samples, features with larger value ranges could outweigh features with smaller value ranges. To prevent this, we use z-score normalization [66] to normalize the feature values before plugging them into the distance functions. Let $\mu_h$ and $\sigma_h$ be the mean and standard deviation, respectively, of the $h^{th}$ feature values for all the data points to be clustered. Then, the $h^{th}$ feature value of a data point (i.e., $x_h$ of a data point $x$) can be normalized as

\[
x_h' = \frac{x_h - \mu_h}{\sigma_h}. \tag{4.23}
\]

To evaluate the performance of these distance functions, we calculate the distances between data points within the same clusters (i.e., events of bird appearance), as well as the distances between data points across different clusters. Figure 4.17 shows the CDFs of the distances between these similar data points and the distances between these dissimilar data points, under each of the three distance functions. For a fair comparison, the distance values are
Figure 4.17: CDFs of distances between similar data (inner cluster) and dissimilar data (inter cluster) under different distance functions.

normalized by using min-max normalization [66]; namely \( d' = (d - d_{\text{min}})/(d_{\text{max}} - d_{\text{min}}) \), where \( d_{\text{max}} \) and \( d_{\text{min}} \) are the maximum and minimum distance values respectively. As we can see, the gaps (areas) between the two CDFs are very similar, which implies that the ability of the three distance functions to measure the similarity/dissimilarity of audio data are quite comparative. Since the gap under weighted Manhattan distance is slightly larger than that under the other two distance functions, we use weighted Manhattan distance in the following experiments.

C. Experiment Results

We evaluate the performance of SimStore by comparing it with the following three schemes.

- **Global-FP**: If all the data points are known before the clustering, then we can use a \( k \)-centering algorithm called Furthest Point [67], which has an approximation ratio of 2. It picks an arbitrary point as the first center, and iteratively find the \((i + 1)^{th}\) center \((i = 1, 2, ..., k - 1)\) as the point that has the maximum distance to the current \(i\) centers selected. In each iteration, it requires one scan of the original data set. Hence, this scheme is not applicable to our problem because the sensory data are collected incrementally along the clustering. It also have been approved [68] that it is NP-hard to approximate the optimal solution with any factor less than 2. So we simply use this scheme as a reference for the performance upperbound.
• **Local-DA**: As described in Section 4.4.2, the Doubling Algorithm can be used to incrementally cluster the data collected on each local node. Without the coordination among nodes, the clustering results on different nodes may overlap and thus give a poor overall clustering.

• **Global-DA**: As described in Section 4.4.3, we can run the Doubling Algorithm in a global fashion on distributed nodes. However, it incurs shifting raw data between nodes and hence introduces extra energy consumption. Therefore, SimStore only enables raw data shifting when energy surplus presents. We use a scheme called Global-DA as a comparison, which is eccentrically SimStore with raw data shifting being always enabled.

We assume that there are totally $N = 9$ nodes in the systems. Each node is continuously collecting audio clips, which we assume are randomly chosen from the 1442 audio clips in our data set. We assume that the local disk of each node can store up to $k$ audio clips, and we test various $k$s in the following experiments.

First, we use BCubed score [69] as a metric to compare SimStore with the above schemes. BCubed score is a commonly used clustering metric when the ground truth is given. Let $\text{Truth}(x)$ be the cluster (a set of cluster members) that a data point $x$ belongs to under the
ground truth. In our case, \( Truth(x) \) is the event that the audio clip \( x \) is associated with. Let \( O(x) \in \mathcal{C} \) be the cluster (a set of cluster members) that \( x \) belongs to in a clustering \( \mathcal{C} \) over a data set \( X \). Then the BCubed score of \( \mathcal{C} \) is defined as

\[
BCubed(\mathcal{C}) = \frac{1}{|X|} \sum_{x \in X} \frac{|O(x) \cap Truth(x)|}{|O(x)|}.
\] (4.24)

The smaller the cluster is and the larger the overlap with the ground-truth cluster is, the higher BCubed score is. A BCubed score close to 1 and 0 imply good and bad cluster respectively. Figure 4.18 shows the BCubed scores of the four different schemes with various \( k = 20, 40, ..., 160 \). Recall that \( k \) is the number of audio clips that can be stored on each node. When \( k = 160 \), the total number of audio clips that can be stored on the system is \( Nk = 1440 \), which is very close to the size of the test set. Therefore, the scores of all the four schemes reach 1 when \( k = 160 \). With the decrease of \( k \), the scores of all the four schemes also decrease. Global-FP assumes that all the data are known in advance, and hence always attains the highest score. Local-DA runs the Doubling Algorithm locally on each node without coordination between nodes, which results in the lowest scores among the schemes. Both Global-DA and SimStore allow nodes to exchange information with the coordinator to assist the clustering in a global view, and thus achieve comparable performance with Global-FP, especially when \( k \geq 80 \). Furthermore, since Global-DA always allow nodes to exchange raw data samples during the clustering, it has a little bit higher scores than SimStore, which only allows raw data exchange when there is energy surplus. As we will see, SimStore reduces the extra energy consumed by exchanging information to a much lower level, while still maintaining very comparative performance with Global-DA.

Second, since the goal of our application is to maximize the number of events that are covered by the stored data, we use event coverage ratio as another metric to evaluate SimStore against the other schemes. We say that an event is covered if at least one of the audio clips stored in the systems is associated with this event. The coverage ratio is the fraction of the
distinct events that are covered by the audio clips in the system. As shown in Figure 4.18, the relative performance of the four schemes is similar to the result with BCubed score. However, for each scheme with the same $k$, the absolute performance is much higher than that with BCubed score. Specifically, SimStore can cover more than 91.1% of the events with $k = 40$, which means that in the bird vocalization collection application, with a storage that can only hold 24.9% data, we can achieve an event coverage ratio as high as 91.1%.

Third, we study the message overhead and energy overhead incurred by Global-DA and SimStore. Global-FP and Local-DA do not rely on node coordination and hence have no
overhead. In contrast, recall that either Global-DA or SimStore relies on exchanging two types of messages. One is the messages to exchange data features between ordinary nodes and the coordinator. The other is the messages to shift raw data between ordinary nodes. Figure 4.20 compares these types of messages used by Global-DA and SimStore. In general, the total number of messages in either scheme decreases with $k$ because less node coordination is needed when the storage space is larger. The total number of messages used by both schemes does not differ too much. However, the number of messages for shifting raw data in SimStore is much smaller than that in Global-DA. The message size for exchanging the 62-dimensional feature values is 1.9 KB, while the message size for exchanging the raw audio clips is 1.3 MB. Therefore, exchanging raw data consumes much more energy than exchanging feature values. Figure 4.21 compares the total energy overhead of Global-DA and SimStore. Since the energy consumed by exchanging feature values is too little to be shown, we do not breakdown the total energy consumption into the two types of messages in the figure. As we can see, the energy overhead of SimStore is much less than that of Global-DA. This implies that SimStore takes only a little amount of extra energy while achieving a performance that is close to Global-DA. Another interesting observation is that the energy overhead does not necessarily decrease along the increase of $k$. This is because when $k$ is
small, it is more likely that all the nodes have no space left in their local storage, and hence less raw data shifting between nodes is triggered.

4.5 Conclusion

In this chapter, we develop a series of storage services under a disconnected system model. Taking into account the renewable and dynamic nature of solar energy, the proposed services adaptively balance between data reliability and data sensing, in order to improve the total amount (utility) of data that can be eventually retrieved from the system.

First, we propose a simple and effective scheme SolarStore, which is independent of physical models of node failure and weather patterns. In spite of the fact that this solution might not be optimal, it guarantees good system performance. Deployment results show that SolarStore dynamically responds to variations in the environment, and has at least 58% less data loss compared to three baseline schemes in different node failure scenarios.

Then, by modeling node failure and weather patterns, we extend the heuristic based SolarStore to a more rigorously formulated problem, and present SolarQoS to provide different qualities of storage for data with different utilities. SolarQoS adjusts the introduced replication factors according to data utility levels and the dynamic energy constraints, with the objective to maximize the total utility of recoverable data from the network. Results show that SolarQoS achieves a total utility that is 48.94% higher than when no replication is used, and 11.27% higher than when data utility difference is not consider.

Furthermore, we extend the static utility model used by SolarStore to a dynamic one, which determines the data utility based on its similarity with other data that have been collected before or on other nodes. We formulate a storage replacement policy as an incremental k-centering problem and propose a distributed scheme called SimStore. Results show that SimStore is very energy efficient and its performance is close to other global schemes that require oracle information of the data being collected.
Chapter 5

Power-based Diagnosis of Node Silence

Troubleshooting unresponsive sensor nodes is a significant challenge in remote sensor network deployments. While prior work often targets low-end sensor networks, this chapter introduces a diagnostic subsystem, geared for remote high-end sensing systems. The main novelty lies in its use of power consumption as a side channel, which has more availability than other I/O ports, to diagnose sensing system failures.

5.1 Introduction

It is common that nodes of dedicated high-end sensor deployments constitute valuable assets such as embedded computers, high-end sensors, as well as expensive energy supplies and storage. For example, our testbed is composed of sensor nodes that cost more than 2,000 each. Therefore, one can easily accrue a total cost that justifies investment in additional diagnostic mechanisms and components.

When remotely-deployed nodes become unresponsive, it is generally hard to determine what caused some node to become silent, without sending a person to the field. If the cost of such field trips is high, remote damage assessment becomes highly desirable to assess the need for intervention. For example, if the cause of the problem is energy depletion (in a solar-powered system), there may not be much that can be done about it until the energy source is restored (e.g., weather improves). On the other hand, if the cause is attributed to a transient error (e.g., a system crash), power-cycling the system remotely may fix the problem. If the cause is attributed to a hardware malfunction (e.g., a radio failure), the
urgency of repair may depend on whether or not the failure has affected the ability of the application to sample and store data. If the application continues to sample and locally store data, then there may be no need for immediate intervention. In contrast, some failures may require urgent attention. For instance, it is urgent to intervene if there is evidence of water damage that may cascade to other nodes or devices. Another example, experienced by the authors on one occasion, was a node that entered a cycle of repeated reboots. The cycle ultimately led to a hardware failure. Early intervention could have saved the node. Our tele-diagnostic system provides strong clues as to what might be wrong with a node, making it possible to plan intervention accordingly.

Prior work on sensor network troubleshooting [70, 71, 72, 73, 74, 75, 76] often focused on bugs that alter system behavior but do not render nodes unresponsive. Hence, a diagnostic system could monitor and communicate node states via regular radio. We cannot use techniques that require the participation of failed nodes since we investigate silent nodes that, by definition, cannot communicate. Some work [77, 78] concerned itself with localizing which node or link failed or malfunctioned, when sensor network performance is impaired. Indeed, while other network nodes can localize a peer’s failure, in this chapter, we take the next step of understanding why the identified peer went silent, as well as retrieving its coarse-grained internal state. Moreover, most of these aforementioned works target low-end sensor nodes (e.g., Tmote and MicaZ motes), while we focus on troubleshooting high-end sensing systems, whose special properties (e.g., relatively high power consumption and high cost) make some solutions that are infeasible for low-end systems become feasible.

When the primary communication channel of a node goes silent due to a failure, a secondary channel is needed to communicate further information on its state. By secondary channel, we mean any mechanism (not necessarily a radio) that conveys or “leaks” information. Several tools [79, 80, 81] were proposed to diagnose target systems using different kinds of I/O ports, such as serial ports or PCI. The applicability of these solutions is, however, restricted by the availability of such ports on the system. Deployed sensor nodes might, for
example, be optimized for power, enclosure cost, or waterproofing ease. Hence, unnecessary I/O ports might be removed. Considering that sensing, communication, computation, and storage necessarily need power, power consumption may be used as a side channel to infer the states of nodes. Compared to other I/O ports, it has a more universal applicability.

Therefore, in this work, we investigate the degree to which power consumption measurements of an unresponsive node can be used as a side-channel to help diagnose the causes of node silence. Specifically, for energy and cost efficiency, we attach an external low-end power meter with its own radio to each deployed node to sample the node’s power consumption, and then the traces of power consumption are wirelessly transmitted to a diagnostic basestation, where they are used to infer the cause of node silence, as well as the health status of the applications on the node.

Our empirical studies show that a high-end sensing node does indeed have a different low-frequency power consumption signature in different normal and abnormal states, leading to the design of our first diagnostic scheme, called **Powertracer**. Before runtime diagnosis, it collects power traces and train a classifier for each possible combination of applications and failure modes, and then uses the classifier to classify the received power traces during runtime. Note that the number of diagnostic states grows exponentially with the number of applications. However, powertracer is still applicable in many deployments which have very specific purposes and thus do not typically run a wide range of different applications concurrently.

In case that a deployment does run a large number of concurrent applications, we propose another diagnostic scheme called **Power Watermarking** that, instead of just passively measures the host’s power consumption, places a module into each host to actively inject unique power patterns (watermarks) into the power consumption traces based on the current system status. Since watermarks adhere to a pre-agreed-upon code, there is no need for prior training. We have implemented both powertracer and power watermarking and compare their performance in this chapter.
The rest of this chapter is organized as follows. Section 5.2 discusses general design guidelines. Section 5.3 presents the implementation of powertracer on SolarStore testbed. Section 5.4 and Section 5.5 explore various diagnostic algorithms based on passive sampling and active watermarking, respectively. The chapter concludes with Section 5.6.

5.2 General Design of Tele-Diagnostic Subsystem

Our objective is to perform remote gross-level damage assessment on unresponsive nodes in high-end sensing systems, such as what may have caused them to stop communicating and what the status of the applications might be. Our design follows two main objectives:

**Diagnostic Subsystem Generality:** It should operate as an external tool, and should be generally applicable to most high-end host systems. Since power consumption is a very general channel that is available on every system that needs power to perform, in this work, we study the possibility of using a power-based tele-diagnostic tool. Moreover, such a tool should require as less changes as possible to the host system. The hardware installation of the tool is simply plugging in a power meter, while one should design its software as simple as possible to make it easier to migrate on different host systems.

**Diagnostic Subsystem Efficiency:** A diagnostic subsystem should not cost, in either components or energy, a sizable fraction of original sensing node cost. Although high sampling frequencies can increase the accuracy of system state estimations, high-frequency Analog-to-Digital Converters (ADCs) are more expensive than low-frequency ADCs, and more energy is required to transmit and process their measurements. Therefore, we aim to devise diagnostic algorithms that can accurately identify node states using meters with low sampling rates.

Following these objectives, we design the tele-diagnostic powertracer as in Figure 5.1. It includes a low-cost power meter, one per sensor node, that periodically samples the current and voltage of its host node. These meters are wirelessly connected via direct, low-bandwidth
Figure 5.1: A power-based tele-diagnostic system as an “add-on” to a sensing system. The watermark generator is needed only when active watermarking is used.

links to a low-end base-station, called the diagnostic base-station, that collects the power traces and runs various diagnostic algorithms to infer the status of unresponsive nodes based on the collected power traces.

Specifically, there are two ways to sample the node’s power consumption. One is passive sampling, which just passively samples the host node without requiring any support from the host. The other is active watermarking, which places a watermark generator (as shown in Figure 5.1) into each host to actively inject unique power patterns (watermarks) into the power consumption traces based on the current system status. These two schemes have their own pros and cons, so we investigate both in the chapter.

Power watermarks can be generated by manipulating the power consumption rate of its host node. For instance, it can alter the CPU load, generate memory or disk operations, or even toggle some peripheral devices. However, for the design objective of generality, the means (e.g., hardware device) chosen for manipulating power consumption should not be specific to a particular sensing system. More importantly, it is desirable for the watermark generator to manipulate the power consumption of a key piece of hardware (e.g., the CPU rather than peripheral devices) that is necessary for application execution as well. This ensures that failure of the watermark generator due to failure of this piece of hardware can
still be correctly interpreted as an indication of application failure, since the application
would then fail as well.

In principle, the availability of independent low-bandwidth wireless communication on
the power meter can also be exploited by the monitored node to send a distress signal if
the node’s main radio fails. We do not exploit it in this work because of the following two
reasons. First, if node failure is brought about by a system crash or energy depletion, having
an extra radio on the failed node would not help as the node would not be able to use it
anyway. Second, and more importantly, it requires connecting the monitored system to the
radio of the diagnostic system, and developing a driver for such a radio on the hardware of
the monitored system, both of which make this solution deeply coupled with the monitored
system and thus violates our design goal of diagnostic subsystem generality.

One should understand that adding a diagnostic subsystem to a remotely-deployed sensor
network, necessarily increases the number of components that are deployed in the field and
hence increases the odds of component failure. The simplicity of the meter, however, where
it merely measures and communicates power samples at a low rate, makes it likely that the
more complex monitored sensing application will fail first. For example, residential power
meters are presently contemplated that should operate uninterrupted for approximately 10
years. Failure of diagnosis, from a user’s perspective, occurs only when both systems have
failed, which has a lower probability than failure of the monitored system alone.

Finally, we emphasize that the diagnostic system, described in this chapter, is intended
to help a remote operator determine the status of deployed, unresponsive nodes. Nodes
that remain responsive can, in general, use other solutions for health monitoring. For ex-
ample, they can run a local diagnostic routine and report its outcome periodically. Such
solutions have been discussed at length in previous literature and hence are not a part of
the contribution of the work presented in this work.
5.3 Implementation on Our Testbed

We deployed the tele-diagnostic subsystem in conjunction with our sensing testbed. Local storage is available on each node to cache the sensory data when the basestation is disconnected. This testbed is a good example of high-end, multi-application sensing systems that our tools aims to troubleshoot.

5.3.1 Hardware

On each sensor node, separately from the above components, the tele-diagnostic powertracer system is installed. Its power meter intercepts the connection between the energy subsystem and the computing subsystem of each node, and reports readings back to a diagnostic base station. As shown in Figure 5.2, the meter is composed of two circuit boards. The first is a custom design that incorporates an Allegro ACS712 hall effect current sensor capable of measuring current consumption of up to 5 amps and an op-amp based difference amplifier to enhance the precision of the meter. The output from the amplifier is connected to an ADC on an off-the-shelf Digi XBee radio, which is itself the second circuit board. The XBee radio we selected is the basic XBee that uses the 802.15.4 protocol and has a 1 mW maximum transmit power. The base station has a matching XBee radio to receive measurements. The meter sample at 1 kHz, and averages the measurements in batches to compensate for the
noise in the measurements. For example, the batch size is 220 by default, which entails an effective sampling rate of 4.5 Hz. By varying the size of sample batches, we can achieve different sampling rates. The entire meter consumes about 871 mWatt, which is only 6% of the host node. The total cost for the parts in each meter is around $59.41, which is about 3% of the cost of the host node (about $2000).

Per our design guidelines, the diagnostic subsystem must be independent from the monitored subsystem. Thus, it is ideal that the energy needed for the power meter itself comes from an independent battery. This is needed to reduce the chances of correlated failures such as energy depletion that causes both the host node and its power meter to fail. However, in our solar-powered testbed, we connect both the meter and the monitored system to the same battery, charged by the solar cell, leveraging the fact that the power meter needs a lower voltage to operate, compared to the monitored system. For example, in our case, the lowest voltage at which the power meter operates reliably is 6.2 Volt whereas the voltage threshold for the monitored system is 10.5 Volt. We connect the meter directly to the battery, bypassing the discharge controller. In this way, the meter continuously reports readings even after the host node is shut down due to energy depletion.

For the sake of this experimental study, we also use the indoor testbed to collect training data for powertracer. Figure 5.3 shows an indoor node with a power meter measuring its power consumption.
5.3.2 Diagnosis Objectives

The objective of our diagnostic subsystem is to determine the cause of connection failures that happen to deployed host nodes, and the current status of the applications that are supposed to be running on those nodes. Namely, hardware and software failures that do not impair the connectivity of a node to the outside world are not the focus of our subsystem. Given a network connection to the node, one can always run all sorts of advanced diagnostic and debugging tools to achieve more fine-grained system diagnosis.

Typically, the common failure causes in remote deployments are known from past experience. For example, our initial experience with our testbed suggests that the most common failure cause is energy depletion. In fact, energy depletion is not an actual “failure” state on this solar-powered platform. No human intervention is needed as a node will eventually come back to live when its energy is restored. However, energy depletion also makes nodes unresponsive, so we need to tell it from other failure cases when it occurs.

Other causes of unresponsive behavior of nodes include operating system crash, infinite loops involving a node reboot, and short-circuit due to water damage (as shown in Figure 4.1). Under those failures, all applications on the failed node would be dead and stop collecting new data. Therefore, it is desirable to make a field trip as soon as possible to repair the node to minimize the data loss and contain the hardware damage.

A node also becomes silent if its Wi-Fi router fails, or the router is still on but the outside antenna is damaged (e.g., by an animal). These two type of failures may not affect the ability of applications to sample and cache data locally. In such cases, there may be no need for immediate intervention, so we can batch multiple such repairs into one trip to save the cost. Therefore, in addition to distinguishing the hardware failures above, we also target identifying the states of applications on unresponsive nodes (i.e., whether or not applications are still functioning correctly and storing data). There are two applications that run on our current deployment perform acoustic and video recording of local wildlife near a
forest; namely, (App-I) collection of bird vocalizations in CD-quality sound, and (App-II) detection of predators of bird eggs using infrared cameras.

To test the accuracy of the diagnostic techniques, we therefore set, as a benchmark, the goal of distinguishing among the twelve failure states shown in Figure 5.4. We take these cases as a proof-of-concept portfolio of failures that we purport to distinguish. In general, as more failures are observed during deployment, and their power traces recorded, they can be added to the portfolio.

Note that, given the 2 applications, there are 4 application subcases that need to be distinguished. In general, if there are $n$ applications that may fail, there is $2^n$ possible application failure states. Therefore, diagnosis based on only passively measured power consumption is not likely to scale to a large number of applications. However, in a sensor network context, it is still useful for many cases of dedicated deployments which often have very specific purposes and do not typically run a wide range of different applications concurrently. For example, only two applications are running in our current deployment.

To handle the cases with multiple concurrently applications, we propose another scheme
called power watermarking. We place a watermark generator into each host node that actively injects unique power signatures (watermarks) into the consumed power traces according to the current system state. At the remote diagnostic basestation, a watermark detector infers the node state by identifying the embedded power watermarks. Since we are only interested in binary per-application states (i.e., running or failed), even with a large number of applications, few bits are enough to code all the possible states of interests. In fact, a simple coding and modulation technique is sufficient for watermark generation, which makes it easier to be implemented on sensor nodes and ported to different systems.

Powertracer has limitation in scalability, but is truly non-intrusive to the host system, while power watermarking is more scalably to the number of applications, but indeed needs to place a simple software module into the host system. In the following, we will present and compare the two schemes in detail.

5.4 Powertracer: Diagnostics Based on Passive Sampling

This section presents an exploration of different design parameters for diagnosing different node failure states from passively measured power consumption traces. The goal is to understand the trade-offs between algorithm complexity and diagnostic accuracy.

As demonstrated in Figure 5.5, we observe obvious difference in the power measurements of our solar-power sensor node under different execution scenarios. This suggests the possibility of using a pre-trained classifier to identify the current system state based purely on the measured power consumption. Specifically, for each system state of interest, we collect the consumed power traces in advance and use the collected labeled data to train a classifier.

When a failure occurs, diagnosing it within a few minutes is considered good enough. We thus use the classifier to determine the state of the system every $\tau$ minutes which we call the detection period. When invoked, the classifier uses a window of size $\delta$ samples to determine
the system state. The following subsections explore the space of possible power trace analysis algorithms from simplest to more complicated, that can be used for classification in order of increasing complexity, as well as hybrid schemes that avoid their individual limitations.

To test the accuracy of the classification schemes, we collected 80,000 samples at the rate of 4.5 samples per second for each of the system states shown in Figure 5.4. We used the first 40,000 samples to extract the static features and the remaining 40,000 samples to test the accuracy of the model. The 40,000 testing samples are divided into traces of length $\delta$, each of which is fed into the classifier to infer the class label. The diagnostic accuracy for a failure case is calculated as the ratio of the number of times that the correct label is identified over the total number of runs.

Our initial experiments show that classifying the power consumption patterns by using static parameters of the probability distribution of the sampled power time-series (e.g., mean and standard deviation) has low accuracy because it does not capture the dynamics of the series. Fortunately, analyzing dynamic time series data and identifying time-varying patterns are very mature research areas in the machine learning and data mining communities. Several techniques that vary in complexity, accuracy and efficiency can be borrowed from the literature [82, 83, 84, 85]. We explore the use of Markov models.

A Markov model determines system states and probabilities of state transitions that best
describe a particular time-series. In this case, we use a simplified version of Markov Models, where the states are predetermined. To build a model for the power trace of a given failure scenario, we process the power trace corresponding to the failure scenario using the following three stages:

**Preprocessing:** As the meter is somewhat noisy, we always perform a outlier filtering step. For collected data for each system state $k$, we first calculate the mean ($\mu_k$) and standard deviation ($\sigma_k$) and discard any value that is outside the range of $[\mu_k - 5 \cdot \sigma_k, \mu_k + 5 \cdot \sigma_k]$ as outlier. Next, we can perform an optional step where we may perform smoothing or normalization to input data based on system configuration (will be elaborated in Section 5.4).

**Discretization:** Power consumption produces continuous-valued time-series data, which are hard to analyze as the possible values for continuous data are infinite. To address this issue, we discretize the power measurements, reducing the numeric values of power samples to a small finite number of symbols. For example, 0-1 Watt can be called an “a”, and 1-2 Watt called a “b”, etc. These symbols represent measured power consumption levels, henceforth called power states. The entire power trace is therefore converted to a string of symbols. Besides this static discretization method, we also examine a dynamic method based on clustering in Section 5.4.

**Computing Transition Probability:** We build a state transition diagram that expresses which states are followed by which other states. For example, a substring “ab” in the power trace string represents a state transition from “a” to “b”. By observing how often “ab” occurs in the string, we can determine the probability of state transition $ab$. For instance, in string “aaabbdcd”, there are total of 6 transitions (e.g., the first “a” is followed by the second “a”, second “a” is followed by the third “a”, third “a” is followed by the “b” and so on). Hence, the transition probability $p(aa)=2/6$ (i.e., there are two transitions from state “a” to “a”), and $p(cb)=1/6$. Any trace can be summarized by a two-dimensional probability matrix that states the probabilities of state transitions from any power state $i$ to
any power state \( j \) in the trace. The aforementioned state transition diagram is also known as a Markov Model. For each system state, we build a model that represents that state.

The models built above are subsequently used for classifying system states during runtime diagnosis. When a node becomes unresponsive, we use the \( \delta \) samples that have just been reported by the power meter. Next, by using the transition matrix of each system state, we calculate the probability that the observed sequence of samples is generated under this model. Specifically, this probability is the product of the transition probability \( p(x_i, x_{i+1}) \) for all \( i = 1, \ldots, \delta - 1 \), where \( x_i \) is the \( i^{th} \) sample in the trace. The system state which has the highest probability of generating the observed sequence is returned as the classification result.

In the following, we conduct a series of experiments to explore the design space of powertracer with respect to different design parameters. For brevity, we refer to the Markov Model as MM.
Figure 5.7: Effect of sampling rate on the classification accuracy of MM (Window size=30 minutes, Number of states=50, Data preprocessing used: Outlier filtering)

Effect of MM Size

To see the effect of the number of MM states on classifier accuracy, we varied the number of states as 5, 10, 15, 20, 30, 40, 50, 60 and 70 and tested the MM with a window size of 30 minutes. The effect of the number of states on accuracy is given in Figure 5.6. For this experiment, we trained the MM on raw data (after noise reduction). As we can see from Figure 5.6, the accuracy increases with number of states and becomes stable after the number of states reaches 50. More interestingly, the figure highlights the fact that increasing the number of MM states far beyond that value is a “bad” idea as well, because accuracy starts to drop if the number of states becomes “too large”. This is because with too many states, the amount of data available in the used window might become insufficient to accurately determine all the state transition probability. In the rest of the section, we use 50 states for MMs unless we specify otherwise.

Effect of Sampling Frequency

Since reducing the sampling interval increases energy consumption, we evaluate the effect of accuracy of the MM classifier with various sampling intervals. We train the MM classifier
Figure 5.8: Effect of window size on the classification accuracy of MM (Number of states=50, Data preprocessing used: Outlier filtering)

at the sampling interval of 222 ms, 444 ms, 888 ms, 1776 ms, 3552 ms, 7104 ms, 14208 ms, 28416 ms, and 56832 ms respectively. The lower sampling intervals were obtained from the same data by down-sampling the original time series (i.e., selecting one every \( N \) original samples for \( N = 1, 2, 4, ..., 256 \)). We present the effect of the sampling interval on accuracy in Figure 5.7. As we can see, if the sampling interval is reduced to 444ms, accuracy starts to drop and after that point the accuracy decreases monotonically due to the loss of information on finer-grained dynamics.

**Effect of Window Size**

To test the effect of window size on accuracy, we trained the MM on the original data (after outliers are removed) with 50 states and tested its accuracy with window sizes of 1, 5, 10, 15, 20, and 30 minutes respectively. Regardless of window size, we considered all windows shifted by 1 minute intervals. We show the effect of varying window size on accuracy in Figure 5.8. In general, increasing window size helps increase the overall accuracy. The amount of improvement varies between different failure states.
Effect of Data Preprocessing

In this section, we consider techniques for data preprocessing that have the potential to eliminate extraneous information from the sampled power signal, allowing us to focus on essential features. For example, the status of a CPU fan ("on" or "off") can affect power consumption by adding or subtracting a constant offset. An MM trained with the fan on may lead to misclassifications if the fan is turned off. To address this problem, we resort to data normalization prior to discretization. We explore two alternatives for normalization; namely, (a) z-score based normalization, and (b) normalization based on relative difference. We describe each of these techniques below.

**Normalization based on z-score:** To normalize the data using z-score, we use the following formula:

\[ x_i' = (x_i - \mu_k)/\sigma_k, \quad (5.1) \]

where \( x_i \) is the raw data, \( \mu_k \) is the mean and \( \sigma_k \) is the standard deviation for the training data for a particular system state \( k \). Intuitively, the z-score represents the distance between the raw score and the population mean in units of the standard deviation. The z-score is
Figure 5.10: Effect of window size on the classification accuracy of MM based on signal difference (Number of states=50, Data preprocessing used: Outlier filtering, difference between consecutive signal)

negative when the raw score is below the mean, and positive when it is above. It is a very common technique for data normalization in data mining literature. In Figure 5.9 we present the impact of varying window size on accuracy of an MM trained based on z-score data. It turns out that the accuracy of MMs using z-score normalization is not encouraging and can not be used for diagnosis effectively.

**Normalization based on difference signal:** As an alternative, we normalize the data using a simpler scheme, that uses the difference signal obtained from the following formula:

\[ x'_i = x_i - x_{i-1}, \]  

(5.2)

where \( x_i \) is the raw data. Note that this scheme is similar to obtaining the derivative of the sampled signal. Hence, any constant bias (such as the power consumption of an irrelevant fan) is eliminated due to differentiation. In Figure 5.10 we present the impact of window size on the accuracy of the trained MM. As we can see from Figure 5.10, the window size has a significant impact on MM classifier accuracy. It has a better classification accuracy compared to the z-score normalization technique, but as good as the MM based on original data. The
intuition behind such poor performance when normalization is used is that because the absolute power consumption level does play an important role in identifying what is running and what is not. Data normalization causes information loss.

**Discretization by Clustering**

Discretization of the power signal is an important step towards computing the MM. In all previous sections, we used a simple discretization technique that simply cut the range of input data into uniform bins and assigned them names. In reality, the power measured for a system in different states may not necessarily be uniformly distributed across its range. The discretization algorithm introduced earlier does not capture nor take advantage of this knowledge. For example, it may put different clusters of power measurements into the same bin. Conversely, there may be bins into which no measurements fall. The first case causes information loss while the latter produces unnecessary states for the MM.

In this subsection, instead of using even ranges, we employ the hierarchical clustering technique [86] to identify representative power levels, and use those representative levels as anchor points to discretize the time-series input. Hierarchical clustering is a common
Table 5.1: Confusion matrix for MM with clustering (Window size=30 minutes, Number of clusters=50, Normalization used: Outlier filtering)

<table>
<thead>
<tr>
<th></th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
<th>f5</th>
<th>f6</th>
<th>f7</th>
<th>f8</th>
<th>f9</th>
<th>f10</th>
<th>f11</th>
<th>f12</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>1</td>
<td>.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td></td>
<td>.94</td>
<td>.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f3</td>
<td>.01</td>
<td>.94</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f4</td>
<td>.14</td>
<td></td>
<td>.86</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>f9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>f10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The technique used in statistical data analysis. It uses a bottom-up approach, where the scheme starts with each data point as a cluster and repeatedly merges each two closest clusters to one until the desired number of clusters is left. Distance between two clusters is given by the average distance between points belonging to the different clusters. In Table 5.1, we show the confusion matrix to determine the probability of misclassification and illustrate which states are most likely to get confused. A cell \((i,j)\) in the confusion matrix represents the probability that of system state \(i\) (the row) will be classified as system state \(j\) (the column). As the results show, the MM classifiers with clustering perform better than earlier schemes. We have 100% accuracy for all the states except \(f3\) and \(f4\). \(f3\) occasionally gets misclassified as \(f4\) and vice versa. It is worth noting that these misclassifications do not affect the ability to recognize which component failed. However, they err in inferring which application is running. This result is encouraging since with the same number of states, the MM classifier with clustering performs better. Figure 5.11 shows the detailed accuracy of each failure state of the system versus the window size.
5.5 Power Watermarking: Diagnostics based on Active Signature Generation

As shown in the results of Section 5.4, powertracer has problem in distinguish some application states (e.g., $f_3$ and $f_4$ in Table 5.1). As the number of applications increases, the number of possible system states grows exponentially making them even more difficult to classify. Although one can expect that the number of applications on dedicated sensing systems is small, to improve scalability, we propose another scheme called power watermarking, which offers a solution by artificially shaping the power trace by adding pre-defined power watermarking. In the following subsections, we present how watermarks are generated on the host node, how they are detected from the noisy power traces that also contains the power variations attributed to all sorts of operations on the host, and evaluate this scheme on our SolarStore testbed.

5.5.1 Watermark Generator

The mission of the watermark generator is to inject power watermarks into consumed power traces to carry system state information. Its coupling with the host node calls for a close look at the original power consumption trace of the host. Figure 5.5(a) shows power measurements of one node, on which App-I is running. App-I continuously samples a microphone and buffers the acoustic samples in the memory. When a predefined size of data has been collected, it processes the raw data into a standard format and then stores it in the local disk. A closer study discloses that those prominent power peaks take place during the raw data processing, which brings the CPU from almost idle to busy\footnote{Note that a CPU generally switches between only two states: busy and idle. The CPU utilization as a percentage is actually an average over a time interval.}.

Based on the above observation, a natural way for generating power watermarks is to alter the CPU state between busy and idle artificially. This approach has the follow advantages.
Algorithm 6 A typical sensing application

```python
while True do
    take samples from sensors of interest;
    process samples;
    sleep(s);
end while
```

First, as shown in Figure 5.5(a), the node power consumption rate jumps from 10.5 Watts, when the CPU is idle, to 13.5 Watts, when the CPU is busy. This difference is substantial enough for most low-end power meters to detect. Second, the CPU is such a key resource that is needed by all applications. Failure of altering the CPU state indicates only one possible system state - the system crash. This ensures that failure of the watermark generation due to failure of this piece of hardware can still be correctly interpreted as an indication of application failures, since the applications would then fail as well. Third, a CPU is generally available on any sensing system, and this makes the watermark generator portable to other systems. Additionally, we observe that other operations (e.g., memory or I/Os) of the applications on the node have non-detectable power changes to the power meter. This is because those operations happen too fast (e.g., within 1 ms) to be detected by low-end power meters, whose sampling rates are usually less than 10 Hz. For other sensing systems where the CPU states are indistinguishable to low-end power meters (e.g., in low-end computing devices), the proposed diagnostic tool is not suitable and not recommended.

In addition to eliminating spurious hardware dependencies of the watermark generator, one should also reduce spurious dependencies on software, preventing software failures of the watermark generator from being falsely interpreted as application failures. Therefore, we propose a decentralized scheme that works as an add-on library to each application, periodically generating heartbeats with a period unique to this application. Since it shares the same address space with its application, it will fail to produce heartbeats if the application crashes. Thus, the absence of particular heartbeats indicates the failure of corresponding applications.
Specifically, since most sensing applications (not only the applications in this work) work in a duty-cycled manner, as shown in Algorithm 6, we have each application call an API provided by the watermark generator in each duty cycle, and thus we can create heartbeats periodically to indicate the health of each application. It is desired that the generation of heartbeats does not interfere with the normal operation of the application that it belongs to. Obviously, a good occasion for generating heartbeats is the time when the application is sleeping. Therefore, our watermark generator provides a new sleep function, which can be called by each application to perform normal sleeping action and has one side capability of generating heartbeats.

We devise the sleep function to generate heartbeats at one designated period chosen by each application. Of course, this designated period can only be kept approximately as the heartbeat generation also depends on how the sleep function is called. Specifically, let $P$ be the designated heartbeat period of an application, and the sleep function attempt to produce a heartbeat of width $H$ at the beginning of each period. When the function is called, it checks the current time $t$ and then decides when to run the dummy code on the CPU to generate heartbeats, when to sleep, and for how long. If the function is called with the sleeping time $s > P$, as shown in Figure 5.12, at $t = 0$, it generates heartbeats at the beginning of each period within $s$, and sleeps for the rest of the time. However, if it is called with a short sleeping time at the time when $(t \mod P) \geq H$, it misses the occasion of generating the heartbeat for the current cycle (e.g., the cycle between $2P$ and $3P$ in Figure 5.12), and hence simply goes to sleep for a duration of $s$. It may also happen that the function is called at time $t$ when $(t \mod P) < H$, it then only produces a partial heartbeat with a length of $H - (t \mod P)$.

However, one should note that the heartbeats observed by the power meter are actually the aggregate CPU usage of both the sleep function and the application. As shown in Figure 5.12, when the sleep function has no chance to produce the heartbeat (e.g., in $[2P, 3P]$), it entails that the application is busy using the CPU and thus makes up the wanted heartbeat.
The same thing happens when only a partial heartbeat is produced by the `sleep` function. This observation inspires our design for the watermark detector, which will be presented in the next subsection.

In order to reduce overlap among heartbeats of different applications, we only use prime numbers as heartbeat periods. Note that, the library interposition technique can be used to intercept the original sleep function calls from applications, so that there is no need to modify the source code of applications to have them call the redefined `sleep` function.

To ensure that the heartbeat generation will not interfere other applications, we devise the `sleep` function such that it switches the scheduling priority of the calling process to the lowest level when it is called, and switches back to the original level when it returns. On Linux, we implement this by using the scheduling system calls `sched_getscheduler` and `sched_setscheduler`. The Linux process scheduler considers two classes of priority when scheduling processes onto the CPU. One is static priority, and the other is dynamic priority. The scheduler will start running processes of dynamic priority only when there is no process of static priority in the TASK_RUNNING state. Therefore, by switching to dynamic priority, the `sleep` function will always yield the CPU to other applications that are running with static priority.
5.5.2 Watermark Detection

On the diagnostic basestation, we continuously collect the samples from the meter attached to each monitored node, and identify the embedded watermarks based on the dynamics of sampled power time-series.

Given a power trace, it is first preprocessed to classify what samples represent the CPU busy state and what samples represent the idle state. A naive way is to measure in advance the power consumption rates for both the CPU states. Instead, we use a clustering-based method that is more resilient to system changes. Since the two power levels for the two CPU states occur very frequently, we observe two masses of samples in the traces, which correspond to the two CPU states. Therefore, we adopt the \( k \)-means (i.e., \( k = 2 \)) clustering method to classify each sample into one of the two classes, \( B \) (CPU busy) and \( I \) (CPU idle).

After preprocessing, a power trace \( X \) of length \( W \) is converted into a series of \( B \)s and \( I \)s. We determine if a particular pattern exists in \( X \) based on its similarity score that is calculated as follows. We denote a candidate heartbeat watermark by \( Y \), and denote the length of \( Y \) by \(|Y|\) (i.e., the period of the heartbeats). Since the candidate \( Y \) may not be synchronized with the trace \( X \), we denote their offset by \( \theta \) where \( 1 \leq \theta \leq |Y| \). The similarity between \( X \) and \( Y \) is calculated as the maximum of the aggregated sample-wise similarity under all possible \( \theta \)s. Formally, we have

\[
\text{sim}(X, Y) = \max_{1 \leq \theta \leq |Y|} \sum_{i=1}^{W} X[i] \odot Y[(i + \theta) \mod |Y|],
\]

where the sample-wise operator \( \odot \) measures the similarity of two samples.

Similar schemes have long been adopted in pattern matching problems, where the operator \( \odot \) is usually defined as the exclusive NOR that returns 1 if the two operands are the same, or 0 otherwise. However, we define our own \( \odot \) based characteristics of our power watermark detection problem. Besides watermarks generated artificially, a power trace also contains noise from applications, back-end system processes and the measurement process. The first two types of noise originate from the CPU usage, so they can only corrupt \( I \)s into \( B \)s, but
not Bs into Is. The noise from the measurement has no bias, and hence it could turn Is into Bs or Bs into Is.

Based on these features of noise, we define the operator \( \odot \) as

\[
x \odot y = \begin{cases} 
1 & \text{if } x = y \\
0 & \text{if } x = B \text{ and } y = I \\
-1 & \text{if } x = I \text{ and } y = B
\end{cases}
\]

By this definition, a sample in the power trace contributes 1 to the similarity score if it matches the pattern. When a \( B \) is wanted by the pattern but an \( I \) is observed in the trace, a penalty \(-1\) is given to the similarity measure, because Bs have immunity to the noise and hence are not supposed to turn into Is. On the other hand, as an \( I \) is pruned to be corrupted into a \( B \), there is neither reward nor penalty when \( x = B \) and \( y = I \).

Let \( S \) be the set of all possible patterns that could occur in \( \mathcal{X} \). Each pattern in \( S \) is the heartbeat pattern with the designated period for each application installed on the host node. If the similarity score of a heartbeat pattern is high enough, we claim that it exists in the trace and hence the corresponding application is still running. Due to the fact that heartbeats with a shorter period (i.e., a higher rate) occur more often within a time window, their absolute similarity score will be higher than that of heartbeats with a longer period. Thus, for a fair comparison, we normalize similarity scores with respect to heartbeat rates. Ideally, a symbol \( B \) should always be produced by either the \texttt{sleep} function or applications at the designated time. It could be corrupted into an \( I \) only by measurement noise. Thus, the zero score gives a very conservative lower bound for declaring existence, which tolerates a measurement error rate of 50%. On the other hand, Is are pruned to be turned into Bs by applications, and hence form some patterns that could be mistaken as heartbeats, resulting in false positives. Thus, in order to obtain a right threshold on the similarity score, we utilize another set of heartbeat patterns, which are not used by any application on the host node and hence are supposed not to exist in the trace. Let \( \mu \) and \( \sigma \) be the mean and standard
deviation of the similarity scores of these non-existing patterns. We use \( \max\{0, \mu + \sigma\} \) as the threshold. With a score above this threshold, a heartbeat pattern is claimed to exist.

### 5.5.3 Evaluation

In this section, we study and compare how accurately the power watermarking can diagnose failures, how it performs when the number of sensing applications increases, and how efficient it is in terms of energy consumption. Since hardware failures can be accurately identified by powertracer, in this section, we will focus on identifying applications states. As shown in Figure 5.4, the application states become unknown only in case of router failure and antenna failure. We define these two types of hardware failures as \( F_1 \) and \( F_2 \), respectively.

In order to test scalability, besides the two applications used earlier, we install 3 other new applications: (\textit{App-III}) an application that logs the output current and voltage of the solar panel; (\textit{App-IV}) an application that monitors the internal temperature of the node; (\textit{App-V}) an application that collects readings from environmental sensors (i.e., temperature, humidity and light sensors). Furthermore, by running multiple instances of each application, we are able to test scalability to a larger state space. For comparison, we also report the overall diagnostic performance of powertracer under each of \( F_1 \) and \( F_2 \) when more than 2 applications are running.

Let \( S \) be the set of prime numbers in the range of \([5, 79]\). We choose \( S_1 = \{5, 11, 17, 23, 31\} \) and \( S_2 = \{41, 47, 59, 67, 73\} \) as the two sets of prime numbers, which are used by the 5 applications as the heartbeat periods (in seconds) for \( F_1 \) and \( F_2 \), respectively. We use the remaining 10 prime numbers in \( S \) to calculate the detection thresholds. For purposes of obtaining an accurate threshold, we choose the two groups of numbers (i.e, \( S_1 \cup S_2 \) and \( S - (S_1 \cup S_2) \)) to be interleaving. The heartbeat duration \( H \) is 1 second, the default sampling rate \( R \) is 10 Hz, and the default window size \( W \) is 900 seconds.

First, we study how heartbeats with designated periods can be identified by using their similarity scores. Figure 5.13 shows the normalized similarity scores of all the periods in \( S \).
for two power traces collected under two different cases, respectively. Note that, the periods in $S - (S_1 \cup S_2)$, denoted by the yellow bars in Figure 5.13, are never used by any application in case of F1 and F2. The mean and standard deviation of the similarity scores of these periods are used to obtain the detection thresholds, which are the red horizontal lines in the figure.

In case (1), the router fails (F1) but all applications are running. This is one of the most complicated cases because we are supposed to identify all the periods in $S_1$. As shown in Figure 5.13(a), the detection threshold is 0.67. The scores of all the candidate periods in $S_1$ are above this threshold. Although the score of the period 67 in $S_2$ is also above the threshold, all other candidates in $S_2$ are below the threshold. The majority existence of candidates in $S_1$ indicates that there is a router failure, and all applications are running. In case (2), the antenna is broken (F2) but all applications are running. As shown in Figure 5.13(b), the detection threshold is 0.06. The scores of all the periods in $S_2$ are above this threshold. Even though some periods, such as 37, 61 and 71, are also above this threshold, we know that these periods are never used. Therefore, we also identify this case correctly.

To investigate the effect of window size on accuracy, we run the detection module with a window size ranging from 100 seconds to 1000 seconds. Figure 5.14 shows the average diagnostic accuracy for F1 and F2 with various window sizes. Overall, increasing the window size helps increase the accuracy, but this also implies a longer response time. As we are
not aiming at instantaneous failure detection, we choose a reasonably large window size $W = 900$ seconds in the following experiments. Moreover, one may observe that F1 has a higher performance than F2. The only difference between these two failures, with respect to power watermarking, is that applications in F1 have smaller heartbeat periods than those in F2. This indicates that accuracy decreases with the increase of period $P$.

As collecting samples can be expensive in terms of consumed power, we present the effect of sampling rate $R$ on accuracy in Figure 5.15. As we can see, if the sampling rate is decreased, accuracy starts to drop, due to the reduction in the number of samples per window, as well
as the loss of information on finer-grained dynamics. However, the achieved accuracy is still above 85% at 5 Hz, which implies that a low-end power meter, efficient in both energy and cost, is sufficient for power watermark detection.

When a new application is installed, we need to assign two new prime numbers to it as its heartbeat periods under F1 and F2, respectively. In addition, we add two more neighboring prime numbers into $S$ to help determine the detection threshold. For example, as shown above, we use 10 prime numbers in [3, 79] as the two sets of heartbeat periods for the 5 applications, and use the other 10 prime numbers left in [3, 79] to calculate detection thresholds. Moreover, for the purpose of obtaining an accurate threshold, we choose two group of numbers which are interleaved. Accordingly, with 10 applications, a period as large as 179 is needed. CPU load, which is about 20% with 5 applications running, also increases monotonically with the number of applications. Therefore, the scalability of this scheme to the number of applications can be evaluated by studying the effect of period length and CPU load.

To study the scalability while isolating the impact of changes in CPU load, we increase the number of applications by running multiple instances of App-III, App-IV and App-V that are not CPU intensive. Figure 5.17 shows that the accuracy of F1 and F2 goes down with the increase of the number of applications. The performance degeneration is mainly

![Figure 5.16: Effect of heartbeat period on accuracy.](image-url)
caused by the errors in detecting heartbeats with large periods. In Figure 5.16, we show the accuracy for heartbeats with different periods for 10 applications. Accuracy drops from 88% for heartbeats of period 5 to 52% for heartbeats of period 167. This is because heartbeat with larger periods appear less within a time window and thus their similarity scores are more vulnerable to noise. Thus, overall accuracy is inversely related to the period. Also, this raises a fairness issue that applications with shorter periods are more favored in diagnosis. One may therefore assign short periods to applications with higher priority in diagnosis. On the other hand, we can see that the diagnostic accuracy of powertracer drops dramatically as the number of applications increases. This is because App-III, App-IV and App-V are not CPU intensive, and thus do not generate obvious power signatures that could be captured by the low-end power meter. Hence, powertracer has problem in distinguishing their states.

To study the effect of CPU load without changing the heartbeat patterns, we still use the 5 applications and their designated periods. A dummy process is used to bring extra load to the CPU. As shown in Figure 5.18, the diagnostic accuracy for F1 and F2 drops with the increase of CPU load, due to the increasing false positives introduced by the dummy application. In fact, CPU load of sensor nodes is not expected to be high as typical sensing
applications are not CPU intensive. This means that our scheme can achieve a reasonably high diagnostic accuracy in most sensor deployments.

Last, we evaluate the energy cost of power watermarking. In Figure 5.19, we show the energy consumption of heartbeat generation with different numbers of applications. It shows the worst case that all applications are alive and generating heartbeats. The Y axis is the ratio of the extra energy consumption brought up by power watermarking over the normal energy consumption of the host node when no power watermarking is used. When the number of applications increases, heartbeats with more different periods are generated. Consequently, as shown in the figure, more energy is consumed. The energy consumption
shown here is normalized by the energy consumption of the host system with no heartbeat, and it becomes relatively small when CPU load increases. In general, the extra energy consumed in this worst case is less than 5% of the host node on our testbed. This percentage is acceptable for high-end sensing systems that our powertracer targets. Recall that we activate heartbeat generation only in case of failure, and this further conserves energy.

As confirmed by the experiment results, power watermarking is a good supplemental to powertracer to identify application states when the number of applications is large. On our testbed, powertracer only has about 12% accuracy in distinguishing application states with the 5 applications, while power watermarking achieves more than 95% accuracy. This is acceptable, as far as scalability goes, considering that the number of applications that run on sensing systems is usually not large. This performance can be further improved if we increase the window size or sampling rate.

5.6 Conclusion

This chapter presented a case and a proof of concept for the use of power as a side-channel for remote diagnostics, where damage assessment is performed on unresponsive, remotely deployed nodes. An independent power-metering subsystem was used to collect power consumption traces of high-end sensor nodes. A number of algorithms were compared in their ability to determine possible causes of failure and infer application states by analyzing the power traces. It is shown that accurate diagnostics is possible by using a pre-trained classifier when the number of states to be classified is not large. This is acceptable for many sensor network deployments where the number of concurrent applications is typically small. To improve the scalability, we also propose a power watermarking scheme, which achieves application diagnosis accuracy of 95% with 5 applications and more than 70% accuracy with 10 applications on our deployed testbed. Power watermarking is a good supplemental to powertracer to identify application states, while it indeed requires to run the watermark
generator inside the host node, which is less non-intrusive than powertracer. The powertracer is currently deployed in conjunction with an outdoor solar-powered high-end sensor system for acoustic and video monitoring. The cost of such a system is likely to decrease given the modern trends in metering hardware.
Chapter 6

Related Work

Driven by the increasing demands from cyber applications for intensive interaction with physical environments, a rich body of work has been conducted to address various related research challenges in building sensing systems and providing sensing services for application needs. In this chapter, we discuss the relevant work to demonstrate the uniqueness of our research.

6.1 Solar-powered Remotely-deployed Sensing Systems

Sensing devices have been utilized in a variety of applications to collect and disseminate data of physical elements. Examples include not only human-centric applications in smart home and urban environments [4, 5, 6, 7, 8], but also applications remotely deployed for environmental and habitat monitoring [11, 12, 13, 14, 16, 17, 18, 19, 20], precision agriculture [9, 10, 87], infrastructure monitoring [23, 24, 88], seismic detection [15, 89], and military surveillance [21, 22].

Previous research work in sensing systems has mainly focused on systems with small-size low-end sensor nodes (e.g., Tmote [34] and MicaZ [35] motes). Thus, the research problems that lots of work aimed to solve root in the systems’ low capabilities in sensing, computing, storage, communication and energy supply. Recently, with the emergence of more and diverse sensing applications, sensing systems with various hardware platforms have been developed
to meet specific application requirements. In particular, systems [27, 90, 91, 92, 93] with high-end computing and sensing capabilities have been deployed for high-bandwidth data collection. In this thesis, we address the challenges in building such high-end remotely-deployed sensing systems, which have not been studied systematically in prior work.

Reliability has long been a critical requirement in deployed systems. Much prior work focused on reliability guarantees that are required on functional [94, 95] or timing [96, 97, 98] behavior. However, in this thesis, we take a data centric perspective, considering collected data as the main output to be protected, hence measuring reliability in terms of the amount of successfully retrievable data.

Because of its availability, power density, and renewable nature, solar energy is usually considered as one of the power sources that are the most suitable for long-term systems deployed in remote areas. A number of works have prototyped the use of solar energy to power sensor nodes [99, 100, 101, 102, 103, 104, 105]. However, most of them have focused on node-level design such as hardware architecture or power control, rather than network-wide or application-level performance optimization such as throughput or reliability. For example, Raghunathan et al. [99] describe the key issues and tradeoffs which arise in the design of a solar energy harvesting system and present a prototype called Heliomote. Kansal et al. [100] and Vigrito et al. [101] have studied how to decide the appropriate working duty-cycle of sensor nodes with information of the energy harvesting. Alippi and Galperti [102] propose a low-energy MPPT (Maximum Power Point Tracker) circuit specifically designed for wireless sensor nodes to optimally convey solar energy into rechargeable batteries. Taneja et al. [103] describe a systematic approach to building micro-solar power subsystems for wireless sensor nodes. Eon [104] is an energy-aware programming language that allows programmers annotates paths in the program with different energy states, while the energy manager adapts these states to current and predicted energy levels.

Recently, there arise a few research efforts dealing with these network-wide metrics in solar-powered wireless sensor networks (WSNs) [106, 107, 108, 109, 110]. For example, Voigt
et al. [106] propose a solar-aware version of Directed Diffusion [30] that preferably routes data via solar-powered nodes. Noh et al. [108] consider the end-to-end delay as a metric of network-wide QoS in solar-powered WSNs. They suggest a low-latency data routing scheme, which considers the information of the harvested energy, deployed location and duty-cycle information about the neighbors. Zhu et al. [109] use ultra-capacitors as the energy storage unit, and design a leakage-aware feedback control technique to decide node duty-cycle to match local and network-wide activity of sensor nodes. However, in this thesis, we focus on reliability issues in building solar-powered remotely-deployed systems, such as end-to-end delivery reliability, in-network storage reliability and failure diagnosis.

6.2 Reliable Data Delivery in Sensing Systems

The problem of how to reliably deliver sensor readings to the basestations has attracted lots of efforts. For instance, Woo et al. [111] point out that routing decisions should exploit link connectivity statistics to achieve reliability. They use a time averaged EWMA estimator to capture such statistics dynamically and maintain routing information in a neighborhood table with constant space regardless of node density. Stann et al. [112] present RMST, a new transport layer for Directed Diffusion, which tracks packet fragments so that receiver initiated requests can be satisfied when individual pieces of an application payload get lost. He et al. [113, 114] borrow the idea of opportunistic routing [115] designed for MANET, and develop a group of reliable data delivery protocols well suitable for sensor network platforms.

The unreliable nature of sensor networks implicitly motivates the use of erasure coding. Researchers have demonstrated some interesting results. Among them, Kim et al. [116] modify and implement erasure codes on Berkeley Mica2 motes. Their experimental records show that each option of information redundancy such as retransmission and erasure codes, can overcome some kinds of failures but suffers from the others. Kumar et al. [117] study FBcast, a broadcast protocol based on the principles of modern erasure codes. Wang and
Wu [118] introduce an erasure coding based flooding scheme that minimizes transmission overhead in flooding. It decides the optimal erasure coding parameters based on its current delivery probability. Wang et al. [119] study the routing performance of using erasure codes in delay tolerant networks.

However, all the aforementioned reliable data delivery strategies, no matter how energy-efficient they are, will trade off energy for redundancy, and thus inevitably impair the lifetime of the system. Different from all the existing works, our solution, SolarCode, which maximally takes advantage of the solar energy surplus, can achieve reliability without compromising the system lifetime.

Erasure coding techniques have several different realizations such as Reed-Solomon codes and Tornado codes [50]. Their tradeoffs are discussed in [49]. In particular, Luby transform codes (LT codes) [49] depend on sparse bipartite graphs to trade reception overhead for encoding and decoding speed. The distinguishing characteristic of LT codes is in employing a particularly simple algorithm based on the exclusive OR operation (⊕) to encode and decode the data. The proposed SolarCode (as well as SolarStore and SolarQoS) in this thesis, in fact, does not rely on the choice of the exact erasure coding algorithm.

6.3 In-network Data Storage in Sensing Systems

Lots of work has been proposed for distributed storage in disconnection-tolerant networks. Much of the work [120, 121, 17, 101] aims at answering query efficiently, balancing load among nodes or maximizing the network storage capacity, but reliability are not considered as an major issue. DALi [120] is a data abstraction layer for providing virtual file system among distributed sensor nodes. It focuses on data organization, search and naming schemes, rather than data reliability optimization. EnviroStore [121] and EnviroMic [17] are also distributed storage systems designed for disconnected operation of sensor networks with the object to maximize the network storage capacity. In [101], Vigorito et al. also propose
a scheme to construct multi-resolution summaries of sensory data and store them in the network for efficient querying. Ratnasamy et al. [122] propose a geographic hash table to store key-value pairs at the sensor node geographically nearest the hash of its key, and data are replicated locally to ensure persistence when nodes fail. Bhatnagar et al. [123] propose a reliable file system for sensor networks, which achieves reliability to some degree through regular backup to neighbors. Dimakis et al. [124, 125] address the problem of how to use erasure codes to enable ubiquitous access to the distributed stored data. However, these reliability schemes are only heuristic based and they do not address the storage or energy allocation in the context of performance optimization. Additionally, most of the storage schemes mentioned here are designed for traditional battery-based systems. Thus, in the aspect of energy usage, they focus on just reducing the consumed energy as possible, which is quite different approach from energy-harvesting sensor networks that our work deals with.

Recently, Wang et al. [126] present an adaptive file system for solar-powered sensor networks. It uses erasure codes for improving storage reliability, but their proposed method is only intuition-based and does not consider multiple levels of data utility. Utility models have been widely adopted in formulating resource allocation problems [127, 128]. In the context of sensory data collection, prior work typically defines utility as a function of the data collection rate and then aims at maximizing the utility of all the data collected under a set of resource constraints. For example, Chen and Sha [129] formulate the problem of data transport in sensor networks as an optimization problem whose objective function is to maximize the amount of information (utility) collected at sinks. They define the utility as a function of the data rate times the end-to-end latency. Su et al. [130] formulate rate allocation problem in sensor networks as a network utility maximization problem by defining utility as a function of the data collection rate. However, in this thesis, we argue that the data utility can also dynamically depend on the data that have been collected in the past or on other nodes. Therefore, our SimStore is a content-aware storage scheme, which dynamically determines data utility based on the similarity of their contents.
Many content-aware schemes have been proposed for data retrieval, caching and distribution on the Internet. VisualSEEk [131] is a prototype system for searching images by their visual features. Guo and Li [63] present new techniques for content-based audio classification and retrieval. They select both perceptual features and mel-cepstral features to measure the similarity of audio data. Aron et al. [132] present an architecture for content-aware request distribution, which distributes the incoming requests to a number of back-end nodes in web server clusters. Jacobson et al [133] propose a new architecture for content distribution and retrieval in content-centric networks.

6.4 Remote System Diagnosis

Reliability of deployed systems has always been a great challenge in wireless sensor network research. Traditional troubleshooting tools, including simulation and emulation [134, 135, 136, 137, 138, 139], are good at pre-deployment testing in a controlled lab environment, where one can afford the luxury of changing and inspecting code, restarting nodes, and replacing broken hardware as necessary [140, 141, 142]. In contrast, troubleshooting deployed systems, especially those deployed in remote outdoor environments [11, 15, 16, 18, 19], is a significantly harder problem.

Quite a few tools were developed to troubleshoot deployed systems remotely. Some of them focus on identifying the cause of anomalous network behavior by discovering which node or link failed or malfunctioned. For example, Sympathy [77] infers node or link failures from reduced network throughput. Memento [78] implements a distributed failure detector by having nodes in the network cooperatively monitor each other. Besides localizing failures, other tools were designed to inspect system states and identify programming bugs [70, 71, 75, 76]. PAD [72] tries to reconstruct a global view of the network from partial information provided by different nodes and identify problems such as software crashes, hardware failures, and network congestion. SNMS [73] can be used to gather systemwide performance statistics.
such as packet loss, and throughput. NodeMD [74] runs a runtime detection algorithm for early detection of failure symptoms. A common underlying assumption of these tools is often that the deployed system is accessible through the normal data channel and can provide information when requested. In contrast, our goal is to infer further information on the unresponsive node, such as whether any applications are still running (albeit are unable to communicate).

When the main communication channel is down, tools are needed to retrieve diagnostic information from side-channels. For example, for the purpose of managing computer servers in a remote data center, out-of-band tools, such as remote supervisor adapters [80] for IBM x86-based servers and iLO [79] for HP servers, have been used. They offer access to servers that are otherwise compromised. These tools are usually based on a System-on-Chip architecture, having their own processor, memory, battery, and network connections. They access the host via I/O ports (e.g., PCI or IPMI). While such ports are ubiquitous in server farms, it is not clear that they are as popular with specialized embedded devices optimized for other deployment considerations such as form-factor, waterproofing, or enclosure cost.

The idea of out-of-band management has also been utilized in telecommunications [143] with the very different purpose of exchanging call control information. In the sensor network community, researchers have also investigated an out-of-band methodology for developing and testing deployed systems. For instance, DSN [81] can be deployed as an additional wireless backbone network for an existing system. A DSN node (e.g., a mote) is attached to each sensor node through a dedicated connection (e.g., via UART ports) to observe, control and reprogram the sensor node.

Compared to the above approaches, power consumption is more general channel that is available on every system that needs power to perform. It has already been demonstrated [144, 145] that power signatures can provide helpful information for the system designer to detect and diagnose problems, but they typically require electric meters with high sampling rates [146, 147]. However, meters with high sampling frequencies are typically too
expensive to be deployed as add-ons to sensor nodes. Besides the cost, more energy would be required to transmit and process their measurements. Fortunately, we are interested only in gross-level assessment of state, to tell whether or not applications are still running. We show that our diagnostic tool can infer such state information using low-end meters with low sampling rates.

Power-based diagnosis is also inspired by security research. In the field of computer security, side-channels have long been pursued for the purpose of obtaining confidential information from computers. Electromagnetic emanations can be a rich source of data [148]. Particular attention has been paid to analyzing the emanations of smartcards, and it has been demonstrated that private keys can be extracted from smartcards by analyzing their power consumption and radiation [149]. Recent smartcard designs explicitly prevent such attacks. Acoustic emanations from certain machines also carry information, although it is not clear if it can be used to compromise cryptographic material without the assistance of a malicious agent on the machine in question [150, 151]. We borrow the idea of exploiting side channels from security research. Favoring simplicity, we investigate the merits of using low-frequency power traces as the side channel.
Chapter 7

Conclusions and Future Work

It is envisioned that Cyber-physical systems will revolutionize how humans interact with the physical world by bridging the cyber and physical spaces efficiently and dependably at all space and time scales. As initial steps towards this goal, sensing devices are increasingly embedded within all kind of physical elements to support the context-aware demands of applications.

In line with the vision of a sensor rich world, in this thesis, we focus on the unique challenges in deploying sensing systems in remote areas, which is one of the indispensable components for the proliferation of worldwide sensor platforms. The challenges include reliable data delivery when an uploading opportunity appears, reliable in-network data storage when the system is disconnected from the outside world, as well as remote diagnosis of unresponsive nodes to assess the urgency of human intervention.

The dynamic nature of the renewable energy source also brings unique challenges in energy management. Unlike in traditional battery-powered sensing systems where energy saving has always been a primary design objective, we show that there is incentive to spend more energy in systems with renewable energy sources. We present the concept of energy surplus, which is resulted from the renewability of the energy source and the finite capacity of energy storage. By dynamically coordinating the energy spending process with the energy harvesting processes, more energy is harvested into the system and hence system performance (e.g., reliability) can be improved by using this extra energy.

We provide a suite of solutions for the above challenges from a data-centric perspective, which regards sensory data as the most valuable output of sensing systems. To mitigate the
data loss in communication and storage, we utilize erasure coding to create redundancy and
dynamically adjust the redundancy level according to the energy availability. To mitigate
the data loss during node silence failures, we explore the feasibility of building a power-based
diagnostic tool from the standpoint of economy, scale and diagnostic accuracy.

Our proposed schemes traverse a wide spectrum of the solution space, ranging from
intuition-based solutions to solutions with guaranteed approximation bounds. Although
the intuition-based solutions such as SolarStore might not be optimal, it has no requirement
on the prior knowledge about the system. On the other hand, if there exist models for
the system behaviors, we can formulate the problems in a rigorous way (e.g., SolarCode
and SolarQoS) and find solutions with guaranteed performance. Therefore, our schemes are
applicable to a variety of scenarios with different system assumptions.

Although we have addressed the major challenges in deploying sensing systems in remote
locations, some problems remain. Since such systems are usually left attended most of the
time, it is very important to prevent the system from getting altered without authorization.
This incurs great demands on system security, not only in the physical world but also in the
cyber world. Moreover, some applications such as infrastructure (e.g., highways or bridges)
health monitoring require guarantied and real-time message delivery, and thus have high
delivery requirements and stringent delay constraints. The connectivity between the remote
sensing system and a basestation could be interrupted, which imposes a huge challenge on
providing qualify of service in intermittently connected networks.

Moreover, we can extend our renewable energy management to other types of computer
systems. With the increase in traditional energy costs and people’s concern for environmen-
tal protection, greenness will be a critical performance criterion. Renewable power (e.g.,
solar and wind power) will be one important energy source to save energy cost and reduce
the carbon footprint. As shown in our work, the primary challenge for renewable energy
management originates from the dynamic nature of the energy source. Adaptive controls
are required to dynamically decide when and how much renewable energy to spend, how
much energy to purchase from the electricity grid if needed, and how much renewable energy to sell back to the power grid if possible. Besides these challenges, renewable energy also brings new opportunities to system redesign. For instance, by leveraging the geographic distribution of the renewable energy, a geographically distributed system may further reduce its total energy cost by shifting work load among its subsystems.

The access and collection of data from large-scale sensor platforms places further challenges to network infrastructure, which has been engineered to support connections between hosts. However, network use in sensing systems is dominated by content distribution and retrieval, and this usage trend is also evolving in other types of systems like social networks. In this regard, we are investigating new content-centric network architectures and try to address the problems in scalability, content fast forwarding and content protection.
References


