Anomalies in Sensor Network Deployments: Analysis, Modeling, and Detection

Giovani Rimon Abuaitah
Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Repository Citation
https://corescholar.libraries.wright.edu/etd_all/746

This Dissertation is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact corescholar@www.libraries.wright.edu, library-corescholar@wright.edu.
ANOMALIES IN SENSOR NETWORK DEPLOYMENTS:
ANALYSIS, MODELING, AND DETECTION

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

By

GIOVANI RIMON ABUAITAH
M.S., Wright State University, 2009
B.S., Birzeit University, 2006

2013
Wright State University
WRIGHT STATE UNIVERSITY
GRADUATE SCHOOL

June 18, 2013

I HEREBY RECOMMEND THAT THE DISSERTATION PREPARED
UNDER MY SUPERVISION BY Giovani Rimon Abuaitah ENTITLED
Anomalies in Sensor Network Deployments: Analysis, Modeling, and Detection BE
ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF Doctor of Philosophy.

______________________________
Bin Wang, Ph.D.
Dissertation Director

______________________________
Arthur Goshtasby, Ph.D.
Director, Computer Science and Engineering
Ph.D. Program

______________________________
R. William Ayres, Ph.D.
Interim Dean, Graduate School

Committee on Final Examination:

______________________________
Bin Wang, Ph.D.

______________________________
Yong Pei, Ph.D.

______________________________
Keke Chen, Ph.D.

______________________________
Shu Schiller, Ph.D.
A sensor network serves as a vital source for collecting raw sensory data. Sensor data are later processed, analyzed, visualized, and reasoned over with the help of several decision making tools. A decision making process can be disastrously misled by a small portion of anomalous sensor readings. Therefore, there has been a vast demand for mechanisms that identify and then eliminate such anomalies in order to ensure the quality, integrity, and/or trustworthiness of the raw sensory data before they can even be interpreted.

Prior to identifying anomalies, it is essential to understand the various anomalous behaviors prevalent in a sensor network deployment. Therefore, we begin this work by providing a comprehensive study of anomalies that exist in a sensor network deployment, or are likely to exist in future deployments. After this thorough systematic analysis, we identify those anomalies that, in fact, hinder the quality and/or trustworthiness of the collected sensor data.

One approach towards the reduction of the negative impact of misleading sensor read-
ings is to perform off-line analysis after storing a large amount of sensor data into a centralized database. To this end, in this work, we propose an off-line abnormal node detection mechanism rooted in machine learning and data mining. Our proposed mechanism achieves high detection accuracy with low false positives. The major disadvantage of a centralized architecture is the tremendous amount of energy wasted while communicating the sensor readings. Therefore, we further propose an on-line distributed anomaly detection framework that is capable of accurately and rapidly identifying data-centric anomalies in-network, while at the same time maintaining a low energy profile. Unlike previous approaches, our proposed framework utilizes a very small amount of data memory through on-line extraction of few statistical features over the sensor data stream. In addition, previous detection mechanisms leverage sensor datasets obtained from an earlier deployment or use synthetic data to test their effectiveness. Our framework, on the other hand, has been entirely implemented in TinyOS as a prototype readily deployable into existing sensor networks, alongside other essential protocols such as sensor data collection protocols. An advantage of our system is the fact that it relies on supervised learning. Supervised machine learning algorithms usually achieve higher accuracy than their unsupervised counterparts given a highly representative common ground truth. Thus, in this work, we also design highly expressive anomaly models that may be leveraged to inject anomalous readings into existing sensor network deployments. In order to do so, we have developed a tool called SNMiner which enables us not only to inject anomalies into a network of sensors, but also to extract important statistical features and evaluate the accuracy of a number of supervised machine learning algorithms.
## Contents

1 Introduction

1.1 Motivations ........................................... 5

1.2 Thesis Objectives and Contributions .................. 6

1.3 Thesis Organization .................................. 8

2 Prevalence of Anomalies In Real World Deployments: The Need For Detection Mechanisms

2.1 Great Duck Island (GDI) ................................. 11

2.1.1 Overview ........................................ 11

2.1.2 Observations ....................................... 12

2.2 Redwood Forests ...................................... 14

2.2.1 Overview ........................................ 14

2.2.2 Observations ....................................... 15

2.3 Intel Berkeley Research Lab (IBRL) .................... 16

2.3.1 Overview ........................................ 16

2.3.2 Observations ....................................... 17

2.4 SensorScope ........................................... 21

2.4.1 Overview ........................................ 21

2.4.2 Observations ....................................... 22
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.3</td>
<td>Feature Collection</td>
<td>142</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Filtering Noise</td>
<td>142</td>
</tr>
<tr>
<td>6.3.5</td>
<td>Classification</td>
<td>143</td>
</tr>
<tr>
<td>6.3.6</td>
<td>Classifier Dissemination</td>
<td>144</td>
</tr>
<tr>
<td>6.3.7</td>
<td>On-line Detection</td>
<td>145</td>
</tr>
<tr>
<td>6.3.8</td>
<td>Re-training / Feature Re-selection</td>
<td>148</td>
</tr>
<tr>
<td>6.4</td>
<td>Aspects of Implementation</td>
<td>149</td>
</tr>
<tr>
<td>6.5</td>
<td>Evaluation</td>
<td>149</td>
</tr>
<tr>
<td>6.5.1</td>
<td>SNMiner Toolkit</td>
<td>150</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Simulation Study</td>
<td>151</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Real-World Experimental Deployment</td>
<td>155</td>
</tr>
<tr>
<td>6.5.4</td>
<td>Analytical Evaluation</td>
<td>156</td>
</tr>
<tr>
<td>6.6</td>
<td>Limitations and Evasions</td>
<td>163</td>
</tr>
<tr>
<td>6.7</td>
<td>Discussion And Future Directions</td>
<td>165</td>
</tr>
<tr>
<td>6.8</td>
<td>Applications</td>
<td>167</td>
</tr>
<tr>
<td>6.8.1</td>
<td>Business Applications</td>
<td>168</td>
</tr>
<tr>
<td>6.8.2</td>
<td>Health-Care</td>
<td>168</td>
</tr>
<tr>
<td>6.9</td>
<td>Summary</td>
<td>169</td>
</tr>
<tr>
<td>7</td>
<td>Conclusions</td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>172</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

1.1 A typical multi-hop sensor network deployment. Intermediate sensor motes are expected to report their own readings, as well as forwarding readings received from other motes towards the base station via communication links. ................................................. 3

2.1 Great Duck Island (GDI) deployment. ................................................................. 12
2.2 Intel Berkeley Research Lab (IBRL) deployment [11]. ................................. 16
2.3 Sensor data for node 15. Missing readings for over 5 minutes are left blank. . 18
2.4 Temperature data of 15 nodes from the IBRL deployment. ......................... 19
2.5 Percentage of sensor data faults in the IBRL dataset. Faults mainly include spike faults, stuck-at faults, and noise faults exhibited by the temperature and humidity sensors as well as clipping faults in the light readings. The number of outliers is negligible. ................................................................. 20
2.6 Number of clipping faults in the IBRL dataset. The histogram shows the number of anomalous readings caused by a clipping fault for each node. .... 21
2.7 Humidity readings spanning 3 days in October. Corrosions forming on the connector to the sensor board on October 6 due to high humidity accounts for the erroneous readings. ......................................................... 23
2.8 Wind speed readings collected from two stations during the Génépi deployment. Data prior to October 19 are unusable due to a software bug in the wind sensor’s driver. ......................................................... 25

3.1 Types of anomalies in sensor network deployments and their corresponding detection mechanisms. ................................................................. 35
3.2 Outlier readings of node 20 in the IBRL deployment. ................................. 47
3.3 Spike faults in different deployments. ............................................................... 49
3.4 Constant faults exhibited by node 6 in the SensorScope Génépi deployment. . 50
3.5 Rain precipitation measured by two nodes (node 4 and 14) in the SensorScope St. Bernard deployment. Node 14 exhibits a noise fault. ......................... 52
3.6 Noise faults in the IBRL deployment. ............................................................... 53
3.7 Clipping faults exhibited by node 20 in the IBRL deployment. The light readings clipped at the highest peak are outside the range of possible ADC values. 56
3.8 Compromise data behaviors during lab deployment. Node 8 emits normal temperature readings and is shown as a reference to compare against compromise nodes (nodes 12, 17, and 20). ................................................. 61
3.9 A demonstration of a sinkhole attack in sensor networks. ............................... 65
3.10 Potential events prevalent in the IBRL deployment. ...................................... 72
4.1 Various modeling techniques for spike faults exhibited by two sensor nodes in the IBRL deployment: nodes 6 and 18. For each node figure, the upper region plots the entire period of the spike fault followed by the steady behavior of a stuck-at fault. The lower region plots the first two hours of the spike period.

5.1 A taxonomy of data mining based anomaly detection approaches in wireless sensor networks

5.2 Clustering-based and SVM-based data-centric anomaly detection in wireless sensor networks

5.3 A scatter plot of the temperature and humidity readings from a 4-hour window of readings (starting March 1) obtained during the IBRL deployment. Anomalies shown in the figure are elliptical anomalies discussed in [103].

5.4 Detecting anomalies with ABANDON

5.5 The detection error of the classifier using all three statistical features (highest gradient, temporal deviation, data range) over all sensing modalities (temperature, humidity, light). False negatives are eliminated at iteration 12. In this specific experimental run, there were no false positives since the very first iteration.

5.6 The first two hours of training data during the experimental deployment. The black lines correspond to readings from normal nodes.

5.7 The first two hours of testing data during the experimental deployment. Acoustic signals exhibited one outlier reading over the 2-hour period (at 3:50 am).

5.8 Accuracy of the ensemble classifier after 100 iterations using decision stump as the base classifier.

5.9 Classification accuracy at 100 iterations. The learning process suffers from over-fitting for larger $T_p$ when $\lambda$ is set too small.

5.10 Accuracy of the ensemble classifier after 100 iterations using decision trees as base classifiers.

6.1 A sensor network deployment leveraging a collection tree-like protocol such as CTP [37]

6.2 Block diagram of the proposed online data-centric anomaly detection framework for sensor network deployments

6.3 Online feature extraction and collection ($\beta = 10, \lambda = 2$)

6.4 Functional block of the SNMiner toolkit

6.5 SNMiner graphical user interface

6.6 Detection errors for different values of $\beta$. Three statistical features were extracted (temporal deviation, data range, and highest gradient). Errors are recorded after 300 boosting iterations.

6.7 Resulting ADTree classifier when $\beta$ is set to 20 during simulation (refer to Figure 6.6). Number of iterations = 65.

6.8 Detection error when extracting all three features and using all three sensing modalities during the experimental deployment of 22 nodes. It takes 7 iterations to achieve a perfect detector.
6.9 The resulting ADTree classifier when all features are extracted over all sensing modalities during the experimental deployment. Boosting stops at iteration 70. (Figure [6.8].)
LIST OF TABLES

1.1 Popular Commercial Sensor Platforms .............................................. 2

2.1 Summary of The Existing Deployments Discussed In This Chapter ........ 30
2.2 Summary of Anomalies Prevalent In The Deployments Discussed In This Chapter 33

3.1 Network Failures, Their Root Causes, and Possible Detection Approaches .... 42
3.2 Node Failures, Their Root Causes, and Possible Detection Approaches .... 46
3.3 Measurement Faults, Their Root Causes, and Detection Features ............. 58
3.4 Malicious Network Behaviors, Attack Characteristics, and Possible Detection Approaches ............................................................ 71

4.1 A Set of All Outliers In The IBRL Dataset Prior To Battery Voltage Drops Under 2.40 Volts ................................................................. 82

5.1 Anomaly Detection Schemes Discussed in This Section ..................... 108
5.2 Summary of Extracted Statistical Features ....................................... 114
5.3 Compromise Data Behaviors (Training Phase) ................................... 122
5.4 Compromise Data Behaviors (Testing Phase) ..................................... 123
5.5 Test Errors of The Evaluated Classifiers ......................................... 127

6.1 A Summary of Anomaly Models ....................................................... 137
6.2 Injected Anomalies During The Training Phase (Simulation) .............. 153
6.3 Injected Anomalies During The Testing Phase (Simulation) ................. 153
6.4 Injected Anomalies During The Training Phase (Real-World Experiment) ... 155
6.5 Injected Anomalies During The Testing Phase (Real-World Experiment) ... 155
6.6 Comparison of Total Complexities Per Node .................................. 161
6.7 Description of Parameters Used In Table 6.6 And Section 6.5.4 .......... 161
ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my advisor and friend Professor Bin Wang. Dr. Wang has been my mentor for six wonderful years. Working together throughout my Master’s led me but to accept a great opportunity of pursuing a PhD under his supervision. He did not only believe in my research ideas, but also helped me take them to the next level, that being through delightful conversations, constructive feedback, and his tremendous assistance in proper formation of my thoughts. I truly owe him for his generous support, his dedication as my mentor, and after all for his trust in me and my work.

Second, I am very grateful to Dr. Yong Pei who supervised my very first sensor networks project, and later gave me the opportunity to be his assistant in mentoring a senior design project. The skills that I acquired when working on these projects were those that minimized the time I spent implementing the prototypes discussed in this dissertation. Dr. Pei has been very caring, patient, encouraging, and always ready to give me an advise on different aspects of my life and career. I always enjoyed the numerous conversations we had about various subjects.

I am also thankful to Dr. Keke Chen for his constructive comments and the time we
spent together discussing improvements and suggestions to the presentation of this research. Dr. Chen is very knowledgeable and I am fortunate to have him on my PhD committee. In addition, I would like to thank Dr. Shu Schiller for being a member of my PhD committee and for her invaluable suggestions pertaining to real-world applications.

My extreme gratitude goes to my father Rimon, my mother Linda, and my brothers Wadie, Rami, and Marco. Them believing in me was my key to completing this dissertation. I remember the days when a single conversation with my father would turn frustration into ambition and fatigue into excitement.

Last but not least, I would like to thank my colleague and friend Dr. Tao Zhang for introducing me to the field of machine learning. I had a great time working with him at the Broadband, Mobile and Wireless Networking Research Lab.
To my parents,

for their love and everlasting encouragement.
Chapter 1

Introduction

Over the last decade, real-life deployments of sensor networks have grown in numbers for various applications. Most of these deployments have helped decision makers better understand their targeted environments by analyzing the data collected from a large number of unattended sensors. A typical deployment of a wireless sensor network (WSN) consists of several tens or hundreds of low-power low-cost tiny devices equipped with sensor boards capable of measuring several phenomena. These devices, usually called sensor motes, are also equipped with wireless communication modules that enable them to communicate with each others. Table 1.1 shows the characteristics of some sensor platforms (i.e. motes) popularly used in sensor network deployments. Two types of communication paradigms are possible: (a) a simplified single-hop communication where every sensor mote communicates directly with a base station. A base station is a more powerful node which may further process and then relay messages towards a connected PC, and (b) a more complex multi-hop communication paradigm that involves forwarding messages from sensing nodes to the base station with the help of other intermediate sensor nodes in the network (see Figure 1.1). The base station node is sometimes called the “sink” while other nodes in the network are referred to as “source nodes.”
Table 1.1: Popular Commercial Sensor Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Image</th>
<th>Manufacturer</th>
<th>Processor</th>
<th>Program/Flash Memory</th>
<th>Data Memory</th>
<th>External Flash</th>
<th>Radio Frequency</th>
<th>Battery Type</th>
<th>OS/VM</th>
<th>Prog. Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>TelosB</td>
<td></td>
<td>MEMSC</td>
<td>8MHz TI MSP430</td>
<td>48Kbytes</td>
<td>10Kbytes</td>
<td>1Mbyte</td>
<td>2.4GHz</td>
<td>TinyOS</td>
<td>nesC</td>
<td></td>
</tr>
<tr>
<td>MICA2</td>
<td></td>
<td>ATmega128L</td>
<td>128Kbytes</td>
<td>4K bytes</td>
<td>8Kbytes</td>
<td>1Mbyte</td>
<td>868/915 MHz</td>
<td>TinyOS, MotoreRunner VM</td>
<td>nesC, Java/C#</td>
<td></td>
</tr>
<tr>
<td>MICAZ</td>
<td></td>
<td>IRIS</td>
<td>8MHz 8bit ATmega1281</td>
<td>116Kbytes</td>
<td>1Mbyte</td>
<td>2.4GHz</td>
<td>2x AA</td>
<td>TinyOS, Contiki</td>
<td>nesC, C</td>
<td></td>
</tr>
<tr>
<td>XM1000</td>
<td></td>
<td>advanticsys</td>
<td>8MHz TI MSP430/2x18</td>
<td>48Kbytes</td>
<td>10Kbytes</td>
<td>-</td>
<td>868/900MHz, 2.4GHz</td>
<td>Rechargeable</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>TinySky</td>
<td></td>
<td>Sentilla</td>
<td>8MHz TI MSP430</td>
<td>48Kbytes</td>
<td>10Kbytes</td>
<td>-</td>
<td>Li-Ion</td>
<td>Squawk VM</td>
<td>Java ME</td>
<td></td>
</tr>
<tr>
<td>SunSPOT</td>
<td></td>
<td>ORACLE</td>
<td>40MHz 32bit Atmel ATMega1280</td>
<td>8Mbytes</td>
<td>1Mbyte</td>
<td>-</td>
<td>3x AAA / Li-Lon</td>
<td>TinyOS</td>
<td>nesC</td>
<td></td>
</tr>
<tr>
<td>Waspnode</td>
<td></td>
<td>Libelium</td>
<td>8MHz ATmega1280</td>
<td>128Kbytes</td>
<td>8Kbytes</td>
<td>2Mbytes + 4Kbytes</td>
<td>868/900MHz, 2.4GHz</td>
<td>None</td>
<td>C++</td>
<td></td>
</tr>
<tr>
<td>Imote2</td>
<td></td>
<td>MEMSC (Intel-licensed)</td>
<td>13-416MHz PXA271 XScale</td>
<td>32Mbytes</td>
<td>256Kbytes + 32Mbytes</td>
<td>-</td>
<td>2.4GHz</td>
<td>TinyOS</td>
<td>nesC</td>
<td></td>
</tr>
<tr>
<td>Sense Core Module 3</td>
<td></td>
<td>Coalesences</td>
<td>4-32MHz 32bit RISC Controller</td>
<td>512Kbytes Shared</td>
<td>128Kbytes Shared</td>
<td>-</td>
<td>Pluggable Resources</td>
<td>TinyOS</td>
<td>C++</td>
<td></td>
</tr>
<tr>
<td>TinyNode</td>
<td></td>
<td>Shockfish</td>
<td>TI MSP430</td>
<td>48Kbytes</td>
<td>10Kbytes</td>
<td>512Kbytes</td>
<td>433 MHz, 868MHz, 915 MHz</td>
<td>Pluggable Resources</td>
<td>TinyOS, nesC</td>
<td></td>
</tr>
</tbody>
</table>

One of the very first successful deployments of sensor networks took place at the Great Duck Island [123]. Researchers used data collected from environmental and occupancy sensors laid inside nests (burrows) to better understand the micro-climate surrounding seabirds, observe weather conditions during the breeding season, and measure the occupancy period of seabirds during incubation. The quality and accuracy of the collected sensor data was a major concern. To achieve higher fidelity in the data, appropriate packaging was necessary to protect the sensors from harsh environmental conditions. Without packaging, batteries may rapidly discharge or sensors may totally fail due to their exposure to chemicals such as carbon dioxide. The latter conditions can result in sensor faults that may highly degrade the quality of the sensor data. Even with the existence of appropriate packaging, sensors might still experience natural unknown faults or calibration drifts impacting the confidence in the generated sensor readings. As another example of an early success of sensor networks is the Redwood Forest deployment [126]. In this deployment, 33 sensor
Figure 1.1: A typical multi-hop sensor network deployment. Intermediate sensor motes are expected to report their own readings, as well as forwarding readings received from other motes towards the base station via communication links.

motes where placed at different heights of a 67-meter tree. Researchers collected temperature readings, relative humidity readings, and readings of solar radiation levels using Photo-synthetically Active Radiation (PAR) sensors to help better analyze the ecological conditions of a tree in the forest. The redwood forest deployment stressed the necessity of designing mechanisms to detect and correct sensor faults in a sensor network due to battery discharges and other possible causes regardless of perfect packaging and pre-deployment calibration.

Sensor networks are also susceptible to malicious threats if deployed in a hostile environment. A malicious threat can vary from a simple passive eavesdropping attack to a total compromise of a sensor node. Sensor nodes, once compromised, can inject falsified readings into the network. Any inconsistent\(^1\) or malicious data behavior as a result from a security threat can degrade the trustworthiness, integrity, and/or accuracy of the sensor

\(^1\)Previous research refers to inconsistent sensor readings that are far from the true value as outliers, anomalous, abnormal, faulty or malicious data. We will use all these terms interchangeably unless clearly stated in this dissertation.
data. The rise of the “Internet of Things” \cite{6} has made WSNs even more appealing to the Internet community by connecting standalone sensor networks deployed in different geographical locations to the Internet and making sensor data accessible to authorized individuals over the Web \cite{1,40,50}. The level of confidence in publicly accessible sensors is still uncertain and there is a huge demand for data cleaning tools to filter out anomalous sensor readings.

To this end, researchers have proposed several approaches to detect inconsistencies and abnormal behaviors that can degrade the accuracy of the sensor data. Among those, approaches rooted in the fields of machine learning and data mining proved to be effective. As part of this dissertation proposal, we explore a number of these techniques, their level of effectiveness, and their weaknesses. After that, we propose two new machine learning, i.e., boosting, based mechanisms for anomaly detection in sensor networks which can identify both sensor data faults as well as compromised data behaviors. The major difference between the two proposed mechanisms is that the first detection mechanism is performed at the base station (centralized) whereas, in the second mechanism, the actual detection runs at the sensor level (online). The latter avoids communicating the large number of anomalous readings to the base station, and hence saving a tremendous amount of energy which is highly proportional to the communication cost. Furthermore, we design a toolkit that allows us to model anomalies and abnormal behaviors in a sensor network. Usually, researchers need to test the accuracy/effectiveness of their proposed anomaly detection algorithms. Anomaly models that are highly expressive are desirable due to the difficulty of obtaining a common ground truth necessary to evaluate against. In addition, with supervised detection algorithms, it is essential to label anomalies according to their type. Highly
representative anomaly models may be injected to eliminate labeling errors during the label-
beling process, or reduce tedious and error-prone efforts imposed by simple visual inspec-
tions of the original dataset. The developed toolkit is also capable of extracting interesting
statistical features and evaluating a number of supervised machine learning algorithms.

1.1 Motivations

This dissertation is driven by five major motives. First of all, a typical sensor network
deployment is highly susceptible to various numbers of anomalous behaviors. Prior to de-
signing a detection or preventive mechanism, it is very crucial to understand the frequency
and characteristics of such behaviors. Once achieving a comprehensive study and analysis
of anomalies prevalent in a sensor network deployment, researchers may focus on a spe-
cific type of anomalous behaviors or rather design a generalized mechanism for detecting a
good number of these anomalies. A number of researchers have identified the existence of
such behaviors over the last decade and have designed detection and/or preventive mech-
anism addressing different types of anomalies. The fact that there is a large number of
such proposed mechanisms imposes possible ambiguity of what is, indeed, being detected.
We break this ambiguity by providing a taxonomy of anomalies prevalent in sensor net-
work deployments and their corresponding detection mechanisms. Second, observing the
anomalous behaviors prevalent in a number of successful sensor network deployments to
date, has motivated us to re-think the existing mechanisms for ensuring the quality of the
collected sensor data. There is a prompt need for anomaly detection mechanisms that iden-
tify highly frequent sensor data faults towards avoiding quality degradation in the collected
sensor data. Third, experimenting with a network of a few sensors at our lab has demon-
strated the feasibility of simulating compromise behaviors in a sensor network with very little effort. A node, once compromised, can emit falsified readings, alter other nodes’ readings, or launch a security attack of any kind. This may render sensor data untrustworthy and unusable by application end-users who seek meaningful rather than trustworthy inferences. Thus, there is also a high demand for anomaly detection mechanisms that can identify and eliminate compromise data behaviors to avoid lack of trust in the collected sensor data and/or data integrity vulnerabilities. Fourth, given the fact that sensor nodes are resource-constrained largely because of the lack of a continuous energy source and the small size of data and program memory spaces, it is crucial to design detection algorithms that take into account such characteristics if such mechanisms are to be deployed at the sensor level. Last, there is a lack of sensor network tools which facilitate the modeling of faults and malicious behaviors in a real-world sensor network deployment. The combination of artificially injected but highly expressive anomalies coupled with real sensor data provide a more realistic sensor network model compared to purely synthetic datasets or publicly accessible anomaly-free datasets. This drives us to develop a GUI tool, we call SNMiner, for that purpose. The development of SNMiner was also indirectly influenced by the lack of a standalone platform for rapid experimentation and evaluation of a machine learning based detection model under examination.

1.2 Thesis Objectives and Contributions

The ultimate objectives of this dissertation are summarized here: (i) Improving sensor data acquisition schemes in order to enhance the quality of the collected sensor data in the presence of various anomalies; (ii) Providing a self-organizing sensor network that takes pro-
active measures in case of fault/compromise detection excluding erroneous sensor readings from calculations; (iii) Facilitating the machine learning process by building a responsive sensor network platform willing to train itself in the face of sensor faults and/or malicious behaviors; (iv) Achieving all of the above while maintaining an energy consumption of the sensor nodes nearly commensurate with that without employing any sort of quality assurance measures.

Our major contributions are summarized as follows. First, we put together a comprehensive study of anomalies prevalent in current and futuristic sensor network deployments. There has been similar studies and surveys in the literature regarding anomalies in sensor networks. However, to the best of our knowledge, up to this moment none has provided a rather complete view of the nature of anomalous behaviors and none has intensively analyzed each of these behaviors. Second, we survey the literature in an attempt to provide a taxonomy of machine learning based anomaly detection techniques in sensor networks. We then propose a centralized data-centric anomaly detection technique borrowing from the field of machine learning and data mining. The technique employs a supervised ensemble learning algorithm called Adaptive Boosting (AdaBoost [35]) which relies on a combination of base classifiers to reduce the classification error and hence improving the accuracy of the ensemble classifier. We have shown that the technique yields a low number of false positives while being able to detect a large number of data-centric anomalies in a sensor network deployment. We further propose an on-line data-centric anomaly detection framework for sensor networks. This framework takes into consideration the high energy consumed by communicating sensor readings all the way to the base station. It solves this issue by running the detection at every sensor node in the network. It also employs an
online feature extraction mechanism whereby simple statistical features are extracted on the fly, and therefore, sensor readings do not need to be stored in memory. As an essential part to every supervised machine learning algorithm, sensor data need to be labeled in advance. However, labeling data by itself is tedious as it usually requires visual inspection of every single original data point. Therefore, as a better alternative, we design highly expressive anomaly models that represent measurement faults in existing real-world sensor datasets, possible events, as well as compromise data behaviors common in hostile deployments. Anomaly modeling/injection, feature extraction, and algorithm accuracy evaluation, all were implemented under a unified toolkit called SNMiner.

1.3 Thesis Organization

The rest of the dissertation is organized as follows. The next chapter emphasizes the need for an anomaly detection mechanism by laying down a number of existing sensor network deployments and identifying instances of anomalies prevalent in such deployments. Chapter 3 provides a taxonomy of anomalies prevalent in sensor network deployments. Corresponding detection mechanisms are also being discussed. In Chapter 4, we design data-centric anomaly models which are highly representative of sensor data faults, compromise data behaviors, as well as possible events. Chapter 5 begins by surveying a number of existing machine learning based data-centric anomaly detection schemes for sensor networks. In the same chapter, we propose a novel data-centric anomaly detection framework also rooted in machine learning which is capable of detecting both measurement faults as well as compromise data behaviors with low false positives. An online version of the later is proposed in Chapter 6. At the end of the same chapter, we present the design goals of the
SNMiner toolkit which was used frequently during this work. Finally, Chapter concludes
the dissertation.
Chapter 2

Prevalence of Anomalies In Real World Deployments: The Need For Detection Mechanisms

Sensor network deployments have enabled automated data collection at a finer granularity compared to human-centric sparse deployments of traditional telemetric data loggers, driving scientists or application administrators to draw better conclusions from the collected sensor data. However, this fine granularity cannot be achieved in the presence of failing nodes, frequent network failures, and/or potentially malicious network behaviors. Moreover, given a perfect and secure network with long-lived sensor nodes, finer-grained data samples may still not guarantee the quality of reasoning during the decision making process unless the sensor dataset is fault-free. Unlike traditional devices, wireless sensor motes are resource-constrained battery-powered devices equipped with few on-board sensors which are susceptible to failures, and batteries cannot always be reliably recharged. Over the years, sensor network deployments have materialized a large number of monitoring and event-detection applications. Examples include monitoring weather conditions [122][126], habitat monitoring and wildlife tracking [5][119][122], monitoring of health conditions, flood detection [50], fire detection [75], volcano monitoring [131], structural health monitor-
ing [61], monitoring the micro-climate inside greenhouses [4], monitoring voltage usages at the micro-grid [22], monitoring water quality [106], measuring the quality of perishable food and medicine [10], and tracing and verification of goods at different phases of the supply chain [10]. Researchers and decision makers have observed several anomalous patterns ranging from functional failures to pervasive sensing faults in most of these deployments. In this chapter, we discuss a few of these deployments to emphasize the need for an anomaly detection mechanism. Towards the end of the chapter, Table 2.1 compares the existing deployments we discuss here while Table 2.2 summarizes the different observations sought in such deployments which revealed a number of anomalies.

2.1 Great Duck Island (GDI)

2.1.1 Overview

The GDI project [122] serves as one of the earliest proof-of-concept sensor network deployments. The main purpose of this deployment was to help ecologists study the distribution and abundance of Leach’s Storm Petrels (a type of seabird species) at the Great Duck Island, Maine. The system deploys a multi-tier architecture (Figure 2.1) where one or more sensor patches are located at the lowest tier and the higher tier forms a transit network of gateways connecting the sensor patches to the remote base station(s). Each sensor patch is comprised of burrow motes equipped with temperature, humidity, and non-contact infrared temperature sensors to monitor the occupancy of nesting burrows, and weather motes to measure the surface micro-climate using temperature, humidity, and pressure sensors. There were two types of deployments for sensor patches (with incremental installments): a single hop network of 49 Mica2Dot [118] motes running TinyOS [124] each of which samples read-
ings every 5 minutes, and a multi-hop network of 98 motes with data sampled every 20 minutes. In addition, a verification network of in-burrow cameras was also deployed to obtain ground truth for occupancy verification by collecting 15-second movies every 15 minutes. The deployment operated over 115 days during the summer and autumn of 2003 and covered an ellipsoidal area of 221 by 71 squared meters. To protect the electronics from severe environmental conditions such as flooding or rain, the internal structure of motes was entirely covered with O-rings and conformal coatings. On-board sensors, however, needed to be exposed to the outside world to preserve their sensing capabilities.

2.1.2 Observations

Data Yield

Data yield is the ratio between the number of data points successfully delivered to the base station and the expected number of data points generated by all nodes in the network. During the multi-hop deployment, several failures (anomalies) were spotted that resulted in a low data yield. For instance, five burrow motes stopped reporting only after 5 days of
the deployment. This was due to rapid depletion of their batteries. Another example was
the failure of gateways or the transit network accounting for a sensor data yield of merely
28%. In addition, the GDI deployment indicated that adequate packaging and efficient
mote placement is essential: only 78 motes were recovered out of 150 devices as some
were moved by animals and most of the recovered motes had their antennas removed or
shortened by animals, eventually causing the node to fail to report its readings.

Measurement Faults

The GDI multi-hop deployment and an earlier deployment of a single hop network in the
year 2002 [122], in which sensor motes also measured ambient light levels and sensor
readings were sampled every 70 seconds, revealed that sensors may frequently malfunc-
tion and provide erroneous or abnormal measurements due to various reasons discussed in
Chapter 3. For example, during the 2002 deployment, about half of the temperature and
humidity sensors reported faulty readings. This was mostly due to water contacts result-
ing in short circuits. A faulty temperature reading manifested itself in a persistent reading
of 0°C, whereas an anomalous humidity reading was either higher than 150% (outlier)
or very small. The temperature sensor most often never recovered while humidity sensors
which reported high spikes recovered after they dried up. Humidity sensors with very small
values were also correlated with permanent node outages: 55% of motes exhibiting this be-
havior failed within 2 days [123]. In addition, wet sensors in general caused the battery
level to drop drastically. Light sensors, on the other hand, reported reliable readings most
of the time with the few exceptions when there was a spike usually correlated with tempera-
ture/humidity faults. Both deployments—single and multi-hop—showed that the operating
voltage threshold is 2.7V when using 2-AA alkaline batteries while using a lithium battery maintained a constant voltage, and sensor calibration, although time consuming, is essential before deployment. Some of these facts have motivated researchers to design new mote architecture [96] which is more solid in face of sensor failures (e.g., cuts the power line to the failed humidity sensor when detected) and lasts longer (e.g., the operating voltage threshold is 1.8V).

2.2 Redwood Forests

2.2.1 Overview

A multi-hop network of Mica2Dot TinyOS-powered sensor motes [126] have replaced traditional data loggers in the redwood forests of Sonoma, California. Motes were attached to the west side of a 70-meter tall redwood tree at approximately 2-meter spacing. Each mote was equipped with temperature, humidity, and two PAR sensors (photo-diodes) to measure the micro-climate surrounding the tree. The network operated over 44 days during the Spring of 2004 and collected readings at a sampling rate of 5 minutes. The collected data helped biologists draw conclusions regarding the moving gradients inside the structure of the tree due to variations of the different phenomena over the tree volume. Packaging included a white skirt cover to protect the internal electronics from water and wind but still exposed the sensors to the environment to preserve their sensing capabilities. To be able to access the data from the outside world, a Stargate [121] gateway was placed at the bottom of the tree which stored sensor readings received over the multi-hop network into a TASK [11] database server.
2.2.2 Observations

Data Yield

The entire network collected only 49% of the total sampled data points. Network and node failures were the underlying reasons behind such a low yield. The gateway node suffered outages at the beginning of the deployment while five nodes stopped reporting during the second week. Researchers of the Redwood Forests deployment emphasized the need for network monitoring tools that can detect and report an abnormal behavior once it occurs.

Measurement Faults

By analyzing the data, researchers have found several anomalous readings. For example, some sensor motes running on a low battery produced erroneous readings where others reported readings outside of the normal range (e.g., humidity readings above 100 %RH). The latter is an example of a clipping fault as will discuss in Chapter 3. Since battery failures accounted for most of the anomalies, a simple anomaly detection and filtration algorithm was applied. Sensor readings, reported when the mote’s voltage was outside the range 2.4V-3.0V, were deleted. In addition, humans were involved to remove motes which reported unexpected readings. This deployment has shown that physical installation (e.g., orientation) of sensor motes is very critical and calibration is inadequate to improve the fidelity of the data and other techniques are needed. A time consuming calibration of the humidity and temperature sensors resulted in only slight improvements.
2.3 Intel Berkeley Research Lab (IBRL)

2.3.1 Overview

Researchers at the Intel Berkeley Research Laboratory (IBRL) deployed an indoor network of 54 Mica2 and Mica2Dot motes in March 2004 [11]. Every 30 seconds, motes collected temperature, relative humidity, light, voltage, and topology information. The intention behind this deployment, which lasted for about 30 days (720 hours), was to verify the capability of the Tiny Application Sensor Kit (TASK) services in keeping the network alive over extended periods of time, and to provide the sensor network community with a publicly accessible sensor dataset allowing developers and other researchers to experiment with. The dataset acts as a benchmark to evaluate and compare anomaly detection algorithms against each others. The deployed motes formed the multi-hop network shown in Figure 2.2. The base station is the node with ID 0 and the farthest nodes at the upper left corner (e.g., node 51) are about 10 hops away from the base station. Sensor nodes ran TinyDB which is part of the TASK sensor kit. TASK also provided visual tools for
inspecting the quality of data and the status of the network.

2.3.2 Observations

Data Yield

The entire IBRL sensor dataset [19] contained 2,313,153 entries pertaining to only about half the expected number of entries. Each entry corresponds to a packet or sample received from one of the deployed sensors. Each packet contains the sampling date, sampling time, sequence number (epoch), node id, temperature reading, humidity reading, light reading, and the voltage level.

IBRL researchers found that 4 nodes failed after the first few days for unknown reasons. Even before failing, a high number of packet loss was observed. For example, for a little more than 8 hours starting 11:24 pm on March 1, packets from node 15 were not delivered to the base station most likely due to network contention or interference. The plot of node 15 data in Figure 2.3 shows missing sensor values over the lifetime of the node. The rest of the motes were able to successfully transmit 30% to 82% of their packets over the lifetime of the deployment. In addition to the previous findings, we observe that the entire network has gone offline for two intermittent days. Although this may seem as a correlation of packet losses across sensors, this is not the case over the entire deployment [11]. The two intermittent days of complete loss may be due to a very high network contention or a base station breakage. The loss of sensor readings, however, seems to be highly correlated (a lost packet would result in total absence of all sensor readings). It is also worth mentioning that in the IBRL deployment, three nodes were potentially deployed later after the very first deployment of the original 54 nodes. They may also have been deployed initially but

\[\text{Missing periods are measured using a time gap of 5 minutes.}\]
Figure 2.3: Sensor data for node 15. Missing readings for over 5 minutes are left blank.

never reported any packets until too late in the deployment time for unknown reasons. It
seems like the three nodes rebooted themselves (or were physically reset) as the sequence
numbers did not always follow a sequential order.

In addition to total loss of packets, some nodes have reported packets with missing
sensor values. Several packets (i.e., samples) from node 5 were delivered to the base sta-
tion intermittently until March 3. However, the sensor readings were completely missing
due to unknown reasons. From node 57, only 3 packets were delivered during the entire
monitoring period, one of which had only a single sensor reading (humidity reading).

Measurement Faults

Rajasegarar et al [104] have analyzed the IBRL set for data anomalies. They found that
node 14 reported few anomalous readings (i.e., anomalous temperature and humidity)
within a four-hour window starting midnight March 1. Within the same time window,
node 37 reported anomalous readings at all times (i.e., anomalous node). It is believed that the reason behind the total anomalies of node 37 is its physical placement close to the kitchen (always higher temperature and humidity) and the unusual interference at the corner where node 14 is located accounts for its slightly erroneous behavior. In addition to anomalies resulted from nodes 14 and 37, other forms of fault were observed in [88]. For example, nodes 24 and 32 as well as other spatially correlated sensor nodes experienced clipping faults in the light readings over the course of the month. Clipping occurred in the middle of the day when the light sensor could not read values exceeding the limits of the analog to digital converter (ADC). As in the redwood forests project, due to low battery levels, some sensors have also shown abnormal behaviors that manifested themselves as spikes followed by a “stuck-at” fault as referred to in [88]. In the latter work, researchers have found that 20% of the total temperature readings were faulty. Figure 2.4 plots the temperature data measured by 15 nodes in the IBRL deployment. The figure reveals a number...
Figure 2.5: Percentage of sensor data faults in the IBRL dataset. Faults mainly include spike faults, stuck-at faults, and noise faults exhibited by the temperature and humidity sensors as well as clipping faults in the light readings. The number of outliers is negligible.

of faulty readings that manifest themselves as spikes, noise, outliers, and stuck-at faults.

We will discuss the characteristics of such faults in the coming chapter. Since temperature and humidity readings are very highly correlated, one can imagine a similar plot for the humidity data. Notice that the majority of the temperature readings are spatially correlated (a node’s temperature reading is close to its neighbors’ readings at a time). Temperature readings are also temporally correlated within a short time window.

Figure 2.5 measures the frequency of faulty readings in the IBRL dataset. A number of sensor nodes (node 18, 29, 30, 32, 46, and 50) can only leverage 47% to 57% of its overall collected temperature and humidity readings. The rest of the readings are faulty and may be harmful to the decision making process. Temperature and humidity sensors exhibit spike, stuck-at, and noise faults whereas light sensors suffer from clipping. Figure 2.6 shows the actual number of clipping faults in the IBRL dataset. About half of the sensor nodes exhibit
Figure 2.6: Number of clipping faults in the IBRL dataset. The histogram shows the number of anomalous readings caused by a clipping fault for each node.

2.4 SensorScope

2.4.1 Overview

The SensorScope system [50], developed at EPFL (Swiss Federal Institute of Technology Lausanne), underwent 7 outdoor deployments over a span of three years (2006-2008) in different locations in Switzerland at a scale ranging from 6 to 97 weather stations. Each weather station is equipped with a well-packaged Shockfish TinyNode [85] and nine environmental sensors which measure a handful of environmental observations every 2 minutes (e.g., air humidity and temperature, soil moisture, rain precipitation, solar radiation, water-mark, and wind direction/speed). Sensor motes receive power from an NiMH battery that is rechargeable via a solar panel located at the top of the station. The two most significant de-
ployments took place at: (i) a rock glacier located at 2500-meter high on top of a mountain in Génépi, Switzerland, and; (ii) the 2400-meter high Grand Saint Bernard pass located between Switzerland and Italy. In both cases, weather data were collected over a multi-hop network of weather stations and transmitted by a GPRS-enabled gateway to a database server. Data are publicly accessible over the Internet and can be viewed using popular Web interfaces (i.e., Google Maps, Microsoft’s SensorMap). The Génépi deployment had 16 stations scattered over 500x500 meters area reporting weather data during August-October 2007 (60 days). Detecting floods and rock falls in the area was the main reason behind this deployment as the mountain is a source of mud streams and rock releases due to the melting of underlying permafrosts. The second deployment consisted of 17 stations deployed in September 2007 for 45 days on a 900-meter long line over the Bernard pass to prevent avalanches [7].

2.4.2 Observations

Data Yield

There were a few missing data points during both the Génépi and Grand St. Bernard deployments. As an example, during the former deployment, station 7 did not report any readings over more than a week during the middle stage of the deployment. Mostly, hardware failures due to short circuits accounted for this behavior. As reported in [3], the total number of data points collected from the two deployments was 5800000 and 4300000 respectively. This results in a data yield of approximately 93% and 87% respectively. In order to be able to monitor the network, SensorScope collected status packets in addition to data packets. Status packets carried energy, network, and topology information that can
help network administrators identify potential network and node failures.

**Measurement Faults**

SensorScope researchers have pointed out several challenges associated with sensor network deployments. Some of the major challenges were: (i) packaging, (ii) calibration, (iii) detecting software bugs during deployment, and (iv) tracing unexpected failures. Sensor packaging is important but difficult. In both deployments, erroneous sensor readings were reported during cold and humid periods due to inadequate packaging. The latter allowed corrosions to form on the connector to the sensor board causing short circuits. This resulted in useless readings from all humidity sensors at Génépi mountain [8] on October 6 as seen in Figure 2.7. It also caused the rain sensor of station 14 at the Bernard pass [50] to report faulty precipitation values. The humidity sensor, a Sensirion SHT75 sensor, was embedded
within a stacked-plates structure and a small adapting circuit. Whereas, the wires were protected using epoxy resin. Calibration is required before, during, and post deployment. After packaging, sensors were pre-calibrated by comparing their values to a reference station over several days. During pre-calibration, some temperature sensors reported an offset of over than $2\,^{\circ}\!C$, which is much higher than the expected offset of $0.3\,^{\circ}\!C$, and needed to be discarded. Once deployed, some sensors such as the wind direction sensor should also be calibrated to avoid significant errors in their measurements. Finally, it is important to re-calibrate the sensors after deployment to flag faulty readings.

Despite the fact that the majority of software bugs are easily detected and fixed before deployment and during indoor testing, some code bugs go undetected especially if configuration parameters are modified right before or during deployment. As an example, by observing the wind speed data of station 6 during the Génépi deployment, developers discovered a subtle bug in the wind sensor's driver: an 8-bit counter was not adequate to store the number of revolutions the anemometer completes during one sampling period. The bug was not detected during pre-deployment testing since the sampling period was set much shorter (30 seconds). This bug affected all wind sensors alike, rendering the wind speed readings unusable over the period during which counter overflows were encountered. Figure 2.8 shows wind speed readings collected from two stations during the Génépi deployment. Clearly, all data before October 19 are useless.

Finally, uncorrelated sensor readings may be reported unexpectedly. In [8], authors joked about one occasion where one of the weather stations continuously generated uncorrelated sensor readings (anomalous node), to find out that it was entirely covered with plastic wraps by construction workers to protect it from a near-by construction.
Figure 2.8: Wind speed readings collected from two stations during the Génépi deployment. Data prior to October 19 are unusable due to a software bug in the wind sensor’s driver.

We have seen that, although sensors were carefully calibrated, anomalies still occurred due to other issues mentioned above (i.e., inadequate packaging, software bugs, unexpected behaviors). In [112], for example, authors have identified SHORT faults (outliers) in three of the 31 weather stations deployed outdoor on the EPFL campus and it is not clear what caused such faults. Flagging and removing anomalies is usually done manually with human intervention. Manual anomaly filtering is time consuming, therefore, SensorScope researchers have called for better approaches that rely on powerful off-line analysis and rapid on-line detection algorithms to filter out outliers and hence to ameliorate sensor data quality.
2.5 GreenOrbs

2.5.1 Overview

The GreenOrbs project \[75,82\] aimed at monitoring the micro-climate and ecological trends in the forest. The collected sensor data support several applications including fire risk evaluation, biodiversity and forestry research, monitoring of carbon dioxide absorbed by trees during the photosynthesis process, and estimation of canopy closure \[82\]. GreenOrbs deployments were carried out at several locations in the forest at a scale ranging from 50 nodes to 349 nodes during a recent deployment \[114\]. The long-term goal is to achieve a year-round kilo-scale sensor network deployment with 1000+ motes. The first deployment operated over 30 days starting July 2008 and consisted of 50 TinyOS-powered TelosB \[96\] motes scattered in a 20,000 meter squared forest at the campus of Zhejiang Agriculture & Forestry University, Hangzhou, China. The deployed motes formed a multi-hop network of 6-hop diameter. The second deployment took place in March 2009 with 120 motes forming a 10-hop network. Since then, the network has been continuously expanded.

The system employs the TinyOS conventional services for network-wide synchronization to enable duty cycling (FTSP \[79\]), reconfiguration (DRIP \[125\]), and sensor data collection (CTP \[37\]). Motes collect temperature, humidity, light, voltage, and carbon dioxide (CO\textsubscript{2}) levels. The network also collects routing, neighborhood, and node statistics information. The latter helps diagnosing the network to identify root causes such as link failures or any functional abnormal behavior in the network \[114\]. Each sensor mote runs on 2 AA batteries and is enclosed inside a plastic box to protect it from harsh environmental conditions such as rain. To maintain light readings close to the true illuminance values, the
top of the box is designed to be transparent, whereas holes on the sides of the box maintain temperature and humidity values close to those outside of the box. Finally, the box is mounted on a bracket to protect it from wild animals.

2.5.2 Observations

Data Yield

At a scale of 330 nodes, GreenOrbs researchers found that the network yield dropped from 60% to 10% when changing the sampling frequency from 20 minutes to 66 seconds [75]. Further investigations revealed that the main reason behind packet losses was packet drops due to poor wireless channels or severe collisions. This accounted for 61.08% of all packet losses. Due to this severe case, about 10% of the nodes were forced to drop a packet 20 to 50 times after 30 failed retransmission attempts. The rest of the packet drops were due to forwarding queue overflows at the receiver side (i.e. severe data congestion). Ingress drops could also take place due to a software bug that leads to a locked memory state of the forwarding queue in CTP. All the ingress drops occurred on less than 5% of the nodes. In addition, a number of nodes never successfully reported data to the sink [75] due to software or hardware problems [80]. Routing loops, on the other hand, were considered silent failures that are hard to detect but can drastically hinder the network performance.

Measurement Faults

Similar to previous deployments discussed above, calibration turned out to be very important but hard and time consuming. For instance, during a deployment test of 21 nodes [82], the maximum deviation of one light reading from another was 6KLux when all motes were placed under the same illuminance level and 8.41KLux when placed under different illumin-
nance levels. Light readings were used to estimate the canopy closure (i.e., the percentage of the ground area vertically shaded by overhead foliage). This shows how important it is to pre-calibrate the sensors before deployment to eliminate instrumental errors and deviations in the light readings between correlated sensors, and hence arrive at an accurate estimation. Despite packaging and calibration efforts, GreenOrbs researchers still identified frequent pervasive sensing faults [75].

2.6 VigilNet

2.6.1 Overview

As one last real-world example, we consider VigilNet, a sensor network deployment for target tracking and surveillance [44]. It is an example of an event-driven application. The major challenge of VigilNet is to accurately estimate the speed and location of an intruder (e.g., a vehicle) while keeping the entire system stealthy and energy efficient. Stealthiness is very important when motes are deployed in hostile environments and therefore, are vulnerable to attacks. The size of a sensor node should be unobtrusively small and the network traffic should be minimized. Positioning sensor motes in hidden areas protects them from node compromise while minimizing network traffic reduces the chance of interception of RF signals and packet sniffing. In VigilNet, 70 Mica2 motes running TinyOS were deployed over a 280-feet long grassy road, 35 motes on each side of the road. The road may typically resemble a hostile area to be monitored. A sensor board is mounted on top of the mote and is capable of measuring magnetic fields generated by the movement of target objects, acoustic signals, motion, and distance of the object. The dual-axis magnetometer can sense slowly moving vehicles at a distance of approximately 8-10 feet. In addition,
few long-range gateways are deployed in the sensor field which are responsible for relaying traffic to a distant base station. The base station is attached to a portable device such as a laptop which, in turn, triggers two cameras in the sensor field when a movement is detected.

2.6.2 Observations

In order to precisely estimate the position of the moving vehicle, the number of faulty readings generated by the magnetometers, due to potential change of power state or hardware failures, should be minimized. VigilNet employs aggregation where a leader node fuses correlated magnetic readings from its group members and discards readings from deviating members. The degree of aggregation (DOA), i.e., the number of group members, is a key parameter which controls the percentage of false positives (i.e., reporting an event while there is no vehicle in the vicinity of the sensor field) and false negatives (i.e., missing reports of moving vehicles). Without employing aggregation, the false positive rate is very high (60%). False positives are eliminated if DOA is set to 3, i.e., a leader node fuses readings from three members before reporting an event to the base station. In addition to on-line fault detection and elimination via in-network aggregation, VigilNet also applies off-line fault detection at the base station by analyzing the spatiotemporal correlation among consecutive events. Consequently, the base station sends power-off signals to shut down misbehaving nodes.

Similar to the deployments discussed previously, VigilNet researchers have also advocated for software calibration to eliminate sensor drifts as much as possible before deployment as well as repeated calibrations over time to adapt to changes in the environment.
Table 2.1: Summary of The Existing Deployments Discussed In This Chapter

<table>
<thead>
<tr>
<th>Name</th>
<th>Num of Motes</th>
<th>Sampling Freq</th>
<th>Mote Type</th>
<th>Duration</th>
<th>Area</th>
<th>Packaging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Great Duck Island</td>
<td>49</td>
<td>5 minutes</td>
<td>Mica2Dot</td>
<td>115 days</td>
<td>221 × 71 m²</td>
<td>O-rings /conformal coating</td>
</tr>
<tr>
<td>Redwood Forests</td>
<td>98</td>
<td>20 minutes</td>
<td>Mica2Dot</td>
<td>44 days</td>
<td>70-meter</td>
<td>white skirt cover</td>
</tr>
<tr>
<td>IBRL</td>
<td>35</td>
<td>5 minutes</td>
<td>Mica2Dot</td>
<td>44 days</td>
<td>70-meter</td>
<td>O-rings /conformal coating</td>
</tr>
<tr>
<td>SensorScope</td>
<td>16</td>
<td>30 seconds</td>
<td>TinyNode</td>
<td>60 days</td>
<td>500 × 500 m²</td>
<td>stacked-plates structure /epoxy resin</td>
</tr>
<tr>
<td>GreenOrbs</td>
<td>17</td>
<td>2 minutes</td>
<td>TelosB</td>
<td>45 days</td>
<td>900-meter</td>
<td>no packaging</td>
</tr>
<tr>
<td>VigilNet</td>
<td>50 - 349</td>
<td>10 minutes</td>
<td>Mica2Dot</td>
<td>30 days</td>
<td>20,000 m²</td>
<td>plastic box</td>
</tr>
</tbody>
</table>

Unlike the deployments discussed above, however, sensor nodes in VigilNet are more vulnerable to security attacks due to the nature of the application. For some experiments, which required a long duration of time, VigilNet researchers could not afford to deploy the system unattended due to security issues [44]. Nodes in VigilNet can also be easily compromised. Despite precise calibration and fault detection via on-line fusion and off-line analysis, node compromise can drastically affect the fidelity and trustworthiness of the vehicle detection estimate. Once a node is compromised, injecting falsified readings in a smart way may harm the aggregation process or even worse these readings become hard to detect off-line. Hence, anomaly detection algorithms should also be able to identify such behavior and potentially eliminate readings from compromised nodes. Furthermore, these algorithms should detect malicious network behaviors that may hinder the network performance (e.g., data yield, latency, etc.) or shorten its lifetime.

2.7 A Smarter Supply Chain

As can be seen, anomaly detection in existing sensor network deployments is a crucial challenge which needs to be addressed. Researchers have also envisioned future deployments at a very large scale. As an example, Franklin et al. [33] have envisioned a supply chain management (SCM) system, an example of a high fan-in (HiFi) system, where sen-
sensor data are initially collected and cleaned at the receptor-level which constitutes a number of field deployable units (FDUs). A FDU has a network of sensors and a set of RFID readers. FDUs may be located at different places in the supply chain, including store shelves, store checkouts, and manufacturing lines. Data are then aggregated and smoothed by Stargate-like devices [121] at the second level (i.e., dock doors level). At the third level (i.e., warehouse level), sensor data are arbitrated and sensor streams are correlated across all devices by full-fledged centralized servers. The interior of the system constitutes the regional centers (fourth level) where data are validated, and the headquarters (fifth-level) at which data mining algorithms are run over the processed sensor data streams on the fly to foster the business decision making process and to provide better sales recommendations.

A HiFi proof-of-concept prototype was proposed by the authors in [33] which constitutes three-levels of the main system’s hierarchy and uses TinyDB [78] for querying as well as TelegraphCQ [13], an adaptive data stream processor. The authors have also demonstrated the significance of pushing data cleaning and data reduction (i.e., aggregation) functionalities to the edges of the HiFi system. This calls for anomaly detection algorithms at the integrated WSN-RFID [109] end. The CSAVA (clean, smooth, arbitrate, validate, and analyze) process discussed in the paper incorporates several data processing tasks at the various levels of the HiFi system: cleaning, fault detection, anomaly detection, calibration, stream correlation, and event monitoring. Cleaning and smoothing are achieved by simple filtering queries; sensor readings with an RSSI (Received Signal Strength Indicator) higher than a certain threshold will be discarded, and if the number of readings over a time window is less than a user-specified threshold, these readings are dropped. This calls for more advanced anomaly detection techniques at the receptor-level which may further improve
accuracy against erroneous and noisy sensor data, detect abnormal sensor activities, and avoid fail-dirty cases where sensor nodes fail to report correct readings due to a possibly drained battery.
<table>
<thead>
<tr>
<th>Name</th>
<th>Data Yield</th>
<th>Anomaly</th>
<th>Observed Anomalies</th>
<th>Impact</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Great Duck Island</td>
<td>28%</td>
<td>gateway outage</td>
<td>main cause of the very low data yield</td>
<td>network failure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>rapid depletion of node’s battery</td>
<td>5 motes failed after 5 days</td>
<td>node failure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>permanent node outages due to very low</td>
<td>55% of nodes failed within 2 days</td>
<td>node failure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>humidity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>broken antennas</td>
<td>only 78 out of 150 motes were recovered</td>
<td>measurement faults</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>outliers, spikes, calibration errors</td>
<td>about half of the temperature and humidity readings were faulty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Redwood Forests</td>
<td>49%</td>
<td>gateway outage</td>
<td>outages at the beginning of the deployment accounted for most of the missing packets</td>
<td>network failure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>rapid depletion of node’s battery</td>
<td>5 nodes failed during the second week</td>
<td>node failure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>low battery, out-of-range/clipping faults,</td>
<td>resulted in several erroneous readings</td>
<td>measurement faults</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>calibration errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBRL</td>
<td>≈ 50%</td>
<td>unknown network failure</td>
<td>4 nodes failed after the first few days</td>
<td>network failure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>network contention or interference</td>
<td>high number of packet loss, nodes transmitted only 30% to 82% of their packets</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>base station breakage</td>
<td>entire network went offline for two intermittent days</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>node reboots</td>
<td>three nodes reported packets too late in the deployment lifetime</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>missing data points</td>
<td>two nodes reported packets with missing sensor data points</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>low battery, spikes, stuck-at faults,</td>
<td>20% of the total temperature readings were faulty, some nodes leveraged only 47%</td>
<td>measurement faults</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>clipping faults, noise, outliers</td>
<td>to 57% of their overall collected temperature and humidity readings, about half of</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the sensor nodes suffered clipping in light readings</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SensorScope (Génépi)</td>
<td>≈ 93%</td>
<td>hardware failure due to short circuit</td>
<td>station 7 did not report readings for over a week</td>
<td>node failure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>stuck-at faults, noise, outliers</td>
<td>all humidity sensors reported faulty readings on October 6</td>
<td>measurement faults</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>calibration errors</td>
<td>temperature sensors reported a calibration offset of more than 2°C</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>software bug</td>
<td>a bug in wind sensor’s driver rendered wind data from all stations useless</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unexpected event</td>
<td>one station covered by plastic wraps continuously generating erroneous readings</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SensorScope</td>
<td>≈ 87%</td>
<td>noise, outliers</td>
<td>faulty rain precipitation values reported by station 14</td>
<td>measurement faults</td>
<td></td>
</tr>
<tr>
<td>(St. Bernard)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GreenOrbs</td>
<td>60% - 10%</td>
<td>packet drops due to link failures</td>
<td>accounted for 61.08% of all packet losses</td>
<td>network failure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ingress drops</td>
<td>all the ingress drops (38.92% of all packet drops) occur on less than 5% nodes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>routing loops</td>
<td>accounted for a good number of silent failures</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>software bugs, hardware problems</td>
<td>a number of nodes never reported data to the sink</td>
<td>node failure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>calibration errors</td>
<td>deviation of 6KLux and 8.4KLux in light readings</td>
<td>measurement faults</td>
<td></td>
</tr>
<tr>
<td>VigilNet</td>
<td>n/a</td>
<td>a moving vehicle</td>
<td>event</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>low battery, hardware failure</td>
<td>false positive rate is very high (i.e., 60%)</td>
<td>measurement faults</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>calibration errors</td>
<td>sensor drifts pre-deployment and over the deployment time</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>security attacks</td>
<td>could not afford to deploy the system unattended</td>
<td>malicious behavior</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 3

A Taxonomy Of Sensor Network Anomalies and Their Detection Approaches

The previous chapter explored several real-world sensor network deployments and identified instances of anomalies in each deployment. In this chapter, we systematically analyze the different types of anomalies prevalent in a sensor network, construct a taxonomy of such anomalies, and discuss potential detection mechanisms necessary to reveal the root causes behind each anomaly. Figure 3.1 illustrates three major types of anomalies that can exist in a sensor network deployment; (i) a natural fault; (ii) a malicious behavior; and (iii) an event. We discuss each category independently, pinpointing the possible root causes and the potential tools or mechanisms necessary to detect (and possibly recover from) all or a subset of these anomalies.

The chapter is organized as follows. Natural faults, malicious behaviors, and events are discussed in the next three sections. Having emphasized the demand for anomaly detection mechanisms in real-world applications and deployments of sensor networks, Section 3.4 summarizes the challenges of designing an anomaly detection algorithm for sensor networks. Finally, we discuss related work in Section 3.5 and conclude in Section 3.6.
3.1 Natural Faults

Natural faults are of high occurrence and are considered major anomalies that need to be addressed in almost every sensor network deployment. They can roughly be divided into:

(a) network failures; (b) node failures; and (c) sensor data faults (i.e., measurement faults).

Network and node failures highly impact the network performance and lifetime and are usually diagnosed either at the base station by specialized tools such as Sympathy [107] and AD (Agnostic Diagnosis) [80] or in a decentralized manner by on-line self-diagnosis tools like the recently proposed TinyD2 [74]. These tools usually attempt to identify the root cause of the failure and resolve the issue immediately either via an automated action
or by human intervention (e.g., changing node location or replacing a dead battery). A comparative survey of diagnosis and debugging tools for wireless sensor networks was compiled in [108]. Visualization tools [53,114] can also be helpful in detecting network and node failures. Sensor data faults, on the other hand, degrade the quality of the overall collected dataset, resulting in imprecise conclusions or maybe meaningless inferences.

3.1.1 Network Failures

Failures inside the network may occur due to several factors including interferences, routing inconsistencies, duplicate transmissions, lack of response from a subset of nodes, or even due to unexpected reasons.

Single Link Failures

In ad-hoc networks, a failing or broken wireless link can be roughly defined as a link with a very poor quality that it becomes impossible to reliably exchange packets between the two nodes at both ends [73].

Root Causes. In a typical deployment, this transient failure is a result of harsh environmental conditions (e.g., wind), node mobility [67], and/or presence of obstacles [60] (e.g., a herd of deers) in the vicinity of the two communicating end nodes. A long-term failure is also possible due to severe packet collisions, congestions at the receiving node, or heavy channel interferences caused by coexisting devices using the same channel. This dramatically increases the packet loss and/or corruption rate over that link.

Detection. Failing links are usually detected by collection or multi-hop protocols [37,97,133] using metrics like WMEWMA (Window Mean with Exponentially Weighted Moving Average) [133], LQI (Link Quality Indicator) [21], ETX (expected transmis-
sions) \[23\], or a four bit link estimation \[32\]. Once detected, the network quickly recovers by allowing a node to use a different link (if there exists one) for data transmissions. Link failures can also be identified by network diagnosis tools upon collecting enough statistics at the base station. In Sympathy, for instance, every node periodically sends connectivity metrics (i.e., node’s routing table and neighbor list), flow metrics (i.e., PACKETS TRANSMITTED counter) as well as channel error metrics (i.e., BAD PACKETS RECEIVED and GOOD PACKETS RECEIVED counters) to the base station \[107\]. The base station also keeps track of few metrics such as a SINK PACKETS RECEIVED counter which indicates the number of received packets from each node. Such statistics help discover disconnected paths to the sink, poor links, and congestions. AD follows a different approach for detecting link failures \[80\]. It exploits the correlation of two metrics (i.e., RetransmitCounter and RadioOnTimeCounter) collected in the same way as in Sympathy. If the correlation score between the two metrics is high, this indicates that most of the radio-on time is spent for retransmitting packets and therefore the link between the node and its parent must be of low quality. Topology visualization tools such as Surge, MViz, and Octopus \[53\] can further show a visual indication of a failure or quality degradation of a link upon tree topology changes.

It is worth mentioning that single link failures usually do not prevent the node from transmitting its sensor readings to the base station. Once the node updates its routing table, neighboring nodes with potentially healthy links to that node will still relay its readings to the sink.
Transient Routing Loops

Root Causes. In distance vector routing, a standard routing mechanism in sensor networks, failure or quality degradation of a single link calls for state updates at each node that needs to avoid the failed link. During state convergence, inconsistencies in routing tables may introduce transient loops [92].

Detection. Loops can be inherently detected by the underlying multi-hop routing protocol. For instance, the data path validation mechanism in CTP [37] allows a node to detect possible loop formations by comparing the path-to-sink cost (ETX) within the received data packet with its own path cost. If the former is smaller, the topology information of the transmitter must be stale, indicating a possible loop. Resolution of transient loops may also take quite a while. As for the current network diagnosis tools, it is difficult to detect routing loop anomalies at the back-end by relying solely on a single metric. AD can detect routing loops by exploiting the correlation between two metrics: (i) LoopCounter which measures the number of times a similar packet has passed a node; and (ii) SuccAckCounter, the number of successful transmissions by a node. If the two are highly correlated, the same packet must have passed through the current node multiple times. Jurdak et al [54] discuss a few approaches to loop detection at a centralized base station (e.g., injecting diagnostic packets into the network, use of dynamic source routing, and maintenance of full topology). However, these approaches do not scale well with the network size.

Broadcast Storms

Root Causes. The broadcast storm problem was first defined in [89] as that of serious duplicates, collisions, and contentions caused by simple flooding and redundant rebroad-
casts. Redundant broadcasts are very common in sensor networks and there is a need for mechanisms to reduce the number of duplicates once detected. As an example, due to link quality degradation, involved nodes in CTP periodically broadcast frequent probing beacons to quickly adjust their routing tables using the adaptive beaconing mechanism. This eventually congests the network with multiple duplicates since no duplicate suppression mechanism is adopted. On the other hand, a node in Trickle [70] (a dissemination mechanism used by other code and parameter dissemination protocols to spread updates in the network) suppresses its own transmission once it hears \( k \) similar summaries from its neighbors. This reduces the number of duplicates to avoid a broadcast storm.

**Detection.** Currently, there is no direct way to tell at the back-end if a broadcast storm has actually occurred. Nevertheless, keeping track of the number of duplicates received by each node may indicate that a storm is more likely to happen.

**Network Partitioning**

This is yet another network failure which is directly associated with link or node failures. Network partitioning is considered a significant failure since in some research work the network lifetime is based on the time till a network partition [14,116].

**Root Causes.** Specifically, a failure of a number of critical nodes or a long-term failure in the link between such nodes and the other parts of the network results in a network partition or cut. Critical nodes are those few nodes that are geographically positioned close to the sinks rather than the network edges. This may be a rare event (i.e., failure) but once occurred, a large subset of sensor nodes can be disconnected. Another form of network partitions are temporal partitions [45] which are a result of displaced duty cycles at
different subnetworks. For instance, one subnetwork is sleeping while the other is actively transmitting readings. The latter is a rare failure if a network-wide time synchronization protocol such as FTSP [79] is in place.

**Detection / Recovery.** Detecting cuts at the back-end is an easy task by keeping track of the number of received packets at each node; lack of packet receptions from a group of sensor nodes over a long period of time indicates a potential network cut. Moreover, as proposed in [115], only a small number of sensor nodes called *sentinels* can be used to detect $\epsilon$-cuts where the parameter $\epsilon$ is the fraction of disconnected nodes and is specified by the end-user. The proposed algorithm can detect network partitions by solely relying on the dead/live state of these few sentinel nodes. To resolve network partitioning, either a human action is required (e.g., repositioning few sensor nodes or replacing batteries of dead nodes) or some other recovery mechanism such as the one proposed by Didi et al [26] in which a mobile node (i.e., a robot) is sent to the location where the partitioning has occurred.

**Silent Failures**

A silent failure [80] is a type of network failures which can go undetected if we solely rely on a single metric. A single metric (e.g., number of packets transmitted) may appear to be stable despite the existence of an anomalous network behavior (e.g., routing loops). Examples of silent failures are ingress drops which occur when a portion of the incoming packets are dropped by the node, and frequent changes of parents in the routing tree. Routing loops can also be considered as silent failures since diagnosis tools may easily overlook such a failure as discussed earlier.

**Detection.** A detection of a silent failure requires diagnostic tools that exploit spatial
and temporal correlations between several metrics periodically collected from each node in the network. For example, AD can detect ingress drops by tracking the correlation scores of two metrics: ReceiveCounter and TransmitCounter. A node may receive packets from its children in the routing tree but denies forwarding some of them. Similarly, frequent parent changes may be detected by AD using the correlation scores between the ParentChange-Counter and the RadioOnCounter metrics. A high correlation score indicates that a node changes its parent almost at every instance when the radio is turned on.

**Other Causes of Network Failures**

Although there are mechanisms to reveal the root causes discussed above, sometimes a network failure is hard to detect. Some sensor network deployments [50, 122] form a network structure that is more complex than the single multi-hop network shown in Figure 1.1. Data packets can go missing due to gateway outages, lost GPRS signals, and failures in other parts of a larger network (e.g., the transit network in the GDI deployment).

Table 3.1 summarizes the different types of network failures discussed in this section. The table shows possible root causes and existing mechanisms for detecting every type of failure.

### 3.1.2 Node Failures

Node failures are another common faults in sensor networks which may severely hinder performance if not rapidly detected and agilely recovered from. A sensor node may fail due to hardware or software crashes, energy depletion, and local rebooting. We briefly outline each of these anomalies (as well as possible root causes) and present potential techniques to detect some of them.
Table 3.1: Network Failures, Their Root Causes, and Possible Detection Approaches

| Type                         | Root Causes                                                                 | Existing Detection Mechanisms |
|------------------------------|RAIN|------------------------------|------------------------------|------------------------------|
| single link failure          | - wind, obstacle, mobility (transient)                                      | - inherently by collection protocols (e.g., CTP [37]) |
| (transient, permanent)       | - severe collisions, congestions, channel interferences (permanent)         | - network diagnostic tools (e.g., Sympathy [107]) |
|                              | - inherently by collection protocols (e.g., CTP [37])                       | - AD [80]                     |
|                              | - network diagnostic tools (e.g., Sympathy [107], AD [80])                   | - topology visualization tools (e.g., Surge, MViz, Octopus [53]) |
| transient routing loop       | - routing table inconsistencies                                             | - inherently by routing protocols (e.g., data path validation in CTP) |
| broadcast storm              | - flooding, redundant broadcasts                                            | - network diagnostic tools (e.g., AD) |
| network partitioning         | - failure of critical nodes                                                 | none                          |
|                              | - permanent link failure between a critical node and rest of network         | - using sentinels [115]        |
|                              | - displaced duty cycles at different subnetworks (temporal partitions)       |                              |
| silent failures (e.g.,       | - a buffer overflow (ingress packet drops)                                  | - network diagnostic tools (e.g., AD) |
| ingress packet drops, parent changes, routing loops) | - old parent-child link degradation (parent changes)                        |                              |
|                              | - routing table inconsistencies (routing loops)                             |                              |

Hardware Failures

Hardware modules of a single node including communication (radio and antenna), computation (CPU), and storage (memory) modules may fail or enter a lock state over the course of deployment due to poor manufacturing or harsh environments (i.e., storms, water floods, etc.). A bad radio chip, a broken antenna [58, 122], or a locked CPU or memory prevents a node, whose sensing modules may still be healthy, from transmitting its sensor data towards the sink and hence rendering such data useless. In other words, the sensor data never leave the node and are never used at the base station for finer-grained analysis.

Software Failures

In spite of the utilization of conventional simulation and hardware emulation tools (e.g., TOSSIM [69], AVR JTAG ICE [81]) for debugging a sensor network before deployment, this combination may still overlook some significant software bugs once the nodes are
pushed to the field and exposed to real-world conditions and dynamic environments. A major bug that leads to a node failure is the one that causes task queue overflows. In addition to task queue overflows, sensor nodes in a real-world deployment may also be prone to operating system (OS) crashes.

**Root Causes.** An example of a task queue overflow is observed in the CC1000 radio stack of TinyOS-1.x where two posts of fairly long execution path of 400 ms are constantly reposted causing other calls to fail. Consequently, the node enters a deadlock state and can no longer transmit readings to the sink.

**Detection.** Detecting software bugs require a source-level debugging tool such as Clairvoyant [138]. Clairvoyant does not require any extra hardware and does not modify the source code on the sensor nodes. It is loaded into the boot sector of each node and is used only when needed. Each node can be debugged separately by using a GDB-like terminal running at the host machine to send typical debug commands (e.g., break, step, watch, backtrace). It is also possible to initiate network-wide debug commands addressing all nodes in the network. The latter commands are spread into the sensor network via simple flooding. Clairvoyant was able to detect the CC1000 bug by sending a *led* logging command to display the value of the head of the task queue on the three LEDs. The head value changes every time a task is successfully posted. It was observed that the LEDs never changed their values and hence concluding a task queue overflow. In addition to source-level debuggers, Agnostic Diagnosis [80] is also capable of detecting the latter bug by correlating two metrics: TaskPostCounter which measures the number of tasks posted by the sensor node and TaskExecuteCounter which is the number of executed tasks. If both metrics are weakly correlated, it is an indication of a potential task queue overflow which causes the node to
eventually fail.

**Lack of Energy Resources**

A power outage may be caused by battery disconnection or lack of energy supplied by the harvesting system in place (e.g., solar panels). If no energy harvesting techniques are employed, the battery eventually dies due to energy depletion. Short circuits due to water contacts (e.g., floods, heavy rain) may also lead to eventual power depletion. Furthermore, although a rare event, a battery may suffer internal failure or it may be disconnected (e.g., loose connection) from the main node circuitry. Full depletion of the energy resource is a very common failure or anomaly in sensor networks as a significant number of deployments use non-rechargeable batteries to power up the sensor nodes [11, 82, 122, 126]. Once the battery is discharged (i.e., battery level is below the operating voltage), a sensor node may not always be able to report its readings successfully [122].

**Node Reboots**

For application purposes, sensor nodes may be configured as desired by the application designer to reboot periodically or there might be cases where a node enters infinite loops or suffers from stack overflows leading to node reboots.

**Detection.** Detecting node reboots by sink-based diagnosis tools is fairly simple. For instance, Sympathy investigates an UPTIME metric of each node. If this counter has rolled over, it indicates a node reboot since a node resets this metric to zero once a node has booted. Furthermore, source-level debuggers [138] may identify a subtle bug which leads to a node reboot. An example of this is a bug in the interrupt routine handler of the Microphone module in TinyOS-1.x. Apparently, because of a constant fire up of the tone detector
interrupt, handler routines are invoked several times eventually leading to a stack overflow. Clairvoyant addresses this bug by sending break, cond, and backtrace commands to the failing node to discover that the stack constitutes of several copies of the same handler routine that lead to continuous node reboots.

Node Failures Detection and Recovery

Detection of a node failure is usually performed by checking the failing node’s neighbors [120]. Sympathy does so by first checking if the sink has received any messages from the failed node over one epoch. If not, the absence of the failed node in the NEIGHBOR LIST of each of its neighbors is an indicator of a total node crash. CTP follows the same approach by checking any stale entries corresponding to the failing node in the routing table of its neighbors. The off-line mechanism proposed by Khan et al [58] can further distinguish among the four root causes of a node failure mentioned above. The proposed approach requires an extra piece of hardware ($60), a power meter, that is attached to every node and acts as a tele-diagnostic power tracer. Each power meter periodically measures the current draw of the line between the battery module and the sensor mote as well as the voltage level. The base station uses the power traces obtained from each meter to train a simple classifier which accepts, as inputs, a number of static statistical features (e.g., mean and variance of power values over a sliding window) as well as a more complex data mining based classifier (Markov Model) to further improve the detection accuracy over the static classifier. Once the training process is complete, classification of every node is performed based on its current power trace reported within a window of the same size. Power profiles of a dead node should exhibit similar failure states and the closest state will be assigned
Table 3.2: Node Failures, Their Root Causes, and Possible Detection Approaches

<table>
<thead>
<tr>
<th>Type</th>
<th>Root Causes</th>
<th>Existing Detection Mechanisms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware failure (bad radio, broken antenna, locked CPU/memory)</td>
<td>poor manufacturing, harsh environmental conditions (e.g., storms)</td>
<td>tele-diagnostic power tracer [58]</td>
</tr>
<tr>
<td>Software failure (e.g., task queue overflow, OS crashes)</td>
<td>constant reposting of multiple tasks with long execution paths</td>
<td>tele-diagnostic power tracer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>source-level debuggers (e.g., Clairvoyant [138])</td>
</tr>
<tr>
<td></td>
<td></td>
<td>network diagnostic tools (e.g., AD [80])</td>
</tr>
<tr>
<td>Lack of energy resource</td>
<td>battery disconnection, battery depletion, water contacts, internal battery failure</td>
<td>tele-diagnostic power tracer</td>
</tr>
<tr>
<td>Node reboot</td>
<td>infinite loop, stack overflow</td>
<td>tele-diagnostic power tracer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>source-level debuggers (e.g., Clairvoyant)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>network diagnostic tools (e.g., Sympathy [107])</td>
</tr>
</tbody>
</table>

as the failure type for that node. Finally, recovery from node failures is inherent in fault-tolerant sensor network protocols. CTP, for instance, can quickly reroute traffic around a failed node in less than a second [37] by applying the adaptive beaconing mechanism.

Table 3.2 summaries the different types of node failures discussed in this section. The table shows possible root causes and existing mechanisms for detecting every specific type of failure.

### 3.1.3 Sensor Data Faults

As mentioned in [38] and [75], network and node failures are functional in that such a failure can drastically impact the network functionality and hence hinder its performance. Whereas, sensor data faults or measurement faults are considered non-functional faults since they only impact the fidelity of the reported data. Measurement faults or anomalies are those readings that deviate from the expected value and can be triggered by a single sensor (e.g., light or temperature sensor) or the entire node can experience a hardware failure or internal malfunction (e.g., loose connection with the sensor board) that drives
(a) Outliers in all sensing modalities (temperature, humidity, and light) seem to be highly correlated

(b) Voltage outliers within the operating voltage range of the temperature and humidity sensor

Figure 3.2: Outlier readings of node 20 in the IBRL deployment.

the node to report false readings for all attached sensors. It is also possible that more than one sensor is attached to the same ADC channel. Thus, a single channel fault will impact all sensors connected to it [106,122]. Sharma et al [113] classify sensor data faults into: CONSTANT, SHORT, and NOISE faults. This naming convention was first suggested by Ramanathan et al [106]. Ni et al [88], on the other hand, have compiled an excellent taxonomy of sensor data faults prevalent in sensor networks. For the sake of consistency with earlier fault classifications, we decide to adapt the same terminologies used in both of these works.

Network diagnosis tools are designed to detect functional faults rather than measurement faults, focusing on data quantity rather than data quality [107]. Therefore, detection of the latter should take a different path. There have been several detection mechanisms proposed in the literature to identify sensor data faults. They range from rule-based mechanisms [106,112] to mechanisms that are based on statistics to techniques rooted in machine learning and data mining. In fact, some of these techniques may be able to detect events or malicious data behaviors and are discussed in both [143] and [136]. Due to the detec-
tion generality of these mechanisms, we discuss a single method [88] for identifying each individual fault. Notice that most of these faults can occur in both a temporal and spatial context. This means that a single sensor may experience an anomalous or faulty behavior within its own data over time, or that fault can be spatially uncorrelated with other sensors attached to neighboring nodes.

**Outlier (SHORT)**

**Characteristics.** An outlier, or a SHORT fault [106,113], is a single isolated sensor reading that is significantly far from the expected reading. This fault has been commonly seen in several sensor network deployments [50,55,106,122] and its cause is usually unknown. In the IBRL deployment, for example, a few nodes exhibited one or a couple of outliers prior to the time where their battery voltage began to drop under the normal operating voltage of the temperature and humidity sensors. Outliers would appear in all sensing modalities (temperature, humidity, and light) concurrently. Figure 3.2(a) shows all instances of correlated outliers within the readings of node 20. It is believed that the correlation between temperature and humidity outliers is due to the fact that temperature and humidity are measured by the same Sensirion sensor (share the same ADC channel). In most cases, sensor data outliers also correlate well with an outlier in the voltage readings. The voltage outlier may be outside the operating voltage range of the sensor. The operating voltage of the temperature and humidity sensor in the IBRL deployment is about 2.35-2.36 Volts. Below this value, the sensor readings become anomalous. Voltage outliers can also lie within the normal operating voltage range as is the case with node 20. Figure 3.2(b) assures that the voltage values for node 20 are in fact outliers as they highly deviate from
The first 6 hours of readings from node 9 in SensorScope St. Bernard deployment. The node tends to recover after a spike.

Node 21 in the IBRL deployment maintains a constant behavior following the spike fault till the end of its lifetime.

(a) The first 6 hours of readings from node 9 in SensorScope St. Bernard deployment. The node tends to recover after a spike.

(b) Node 21 in the IBRL deployment maintains a constant behavior following the spike fault till the end of its lifetime.

Figure 3.3: Spike faults in different deployments.

the voltage values in previous epochs. Other outliers may appear in the IBRL dataset but only within a period of a spike fault as we will discuss shortly.

Detection. Typically, the detection of this fault is the process of examining the rate of change between two successive readings, if the change is dramatically higher, then an outlier or SHORT fault has just occurred. Since outliers are useless/insensitive data which is highly uncorrelated with the underlying phenomenon, it can simply be discarded by the sensor network application.

Spike Faults

Characteristics. Similar to an outlier, a spike is also a reading highly deviated from the expected reading. However, a spike spans over more than one reading consecutively and may or may not return to the normal behavior afterwards. Spike faults are very common in sensor network deployments. Figure 3.3(a) shows two instances of a spike fault exhibited by node 9 in the SensorScope St. Bernard deployment during the first 6 hours of its lifetime. Another example of a spike fault is seen in a few soil sensors that measured the
ammonium concentration in a 48-sensors 12-day Bangladesh deployment \cite{88, 106}. This deployment took place in 2006 and aimed at finding the amount of arsenic in groundwater. In both examples, the data returned to its normal behavior after the spike fault. In IBRL dataset, spike faults are more frequent and have higher variance and gradient. Moreover, in the IBRL data, most nodes begin exhibiting a spike fault when approaching the end of the deployment (between March 19 and March 25) as their battery voltage level goes under 2.36 Volts (see Figure 3.3(b)). Node 18 is the only node that loses its battery power much earlier than the other nodes and begins exhibiting spike faults on March 8 (see Figure 3.6(a)). After a number of epochs ($\approx$ 2910 epochs or 1 day), a spike fault transforms to a stuck-at fault where the sensor readings get stuck at a constant value as we will discuss shortly.

Low battery, failure of the sensor module, or a short-circuited connection are potential causes of a spike fault. Apart from being a fault, depending on the application, spikes
can also be a reflection of the real phenomenon. For instance, light sensors generate large spikes when a room becomes suddenly dark [71, 129].

**Detection.** A trivial way to detect this fault involves modeling the rate of change over a short period of time (i.e., within a pre-defined window size). Distinguishing between an anomalous event and a spike fault is a difficult task as we will discuss later in Section 3.3 Therefore, it is not recommended to discard spike faults unless further examination is carried out to find whether it is a result of a potential failure or a true behavior of the measured phenomenon.

**Stuck-At (CONSTANT) Faults**

**Characteristics.** As the name implies, a “stuck-at” fault is a series of readings that maintain steadiness for the rest of the node’s lifetime or may return back to normal after a significant amount of time. False readings may be in or outside the range of the measured phenomenon. Indeed, the latter is what Sharma et al [113] refer to as a CONSTANT fault. The fault has been spotted in a few deployments (e.g., IBRL [11], NAMOS [3], and SensorScope [50]). The rapid 24-hour deployment of the Networked Aquatic Microbial Observing System (NAMOS) took place in August 2006 and consisted of 9 motes (buoys) located at Lake Fulmor, James Reserve. The motes reported temperature and chlorophyll concentration every 10 seconds. Figure 3.4 shows an example of stuck-at faults exhibited by the temperature sensor of node 6 in the SensorScope Génépi deployment. Notice that the faulty readings are outside the range of the expected temperature. On the other hand, referring back to Figure 2.7 we can see that some of the humidity sensors exhibited a constant fault where the faulty readings were still considered in the range of the true phenomenon.
Figure 3.5: Rain precipitation measured by two nodes (node 4 and 14) in the SensorScope St. Bernard deployment. Node 14 exhibits a noise fault.

In the IBRL deployment, almost all nodes exhibited a constant fault over their course of operation. Instances of a stuck-at fault are shown in Figure 3.3(b) following a spike fault.

Detection. Typically, variance is used as the main detection mechanism since a near-zero variance within a time window of a predefined size represents a constant behavior. As the case with spike faults, “stuck at” anomalies should not be discarded until we are certain that it is a case of failure (usually a sensor ADC malfunction) rather than an exhibition of the true phenomena (as we will see in the case of an expected clipping fault in the light data when we discuss out-of-range/clipping faults).

NOISE Faults

Characteristics. This fault was similarly labeled by both Sharma et al. [113] and Ni et al. [88]. In the former, it refers to high variance in the data and the data are usually highly uncorrelated with (distant from) the expected readings. The latter also emphasized
(a) Node 18 exhibits a noise fault at the very end of the deployment when the battery level becomes very low.

(b) Noise fault in temperature and humidity readings of node 6. Both faults are not correlated as is the case with temperature and humidity outliers.

Figure 3.6: Noise faults in the IBRL deployment.

The existence of highly variant data which can still resemble the true physical phenomenon. Henceforth, it is important to distinguish between an expected noise which is within the range of the expected behavior and a definitive fault due to low battery or a potential hardware failure (short circuit). One example of a noise fault was spotted at the SensorScope St. Bernard deployment within the rain meter readings of node 14. As can be seen in Figure 3.5, during the noise fault, readings have high variance. In the IBRL deployment, we consider noise to be the fluctuation of highly deviated constant values that appear at the end of the node’s lifetime when its battery is drained. This behavior is mixed with a stuck-at behavior as seen in Figure 3.6(a). Only few number of nodes (nodes 6, 18, and 23), in fact, produced highly noisy data. Figure 3.6(b) shows a closer look at the largest noise fault exhibited by node 6. The temperature reading rapidly fluctuates between two values 1.1731°C and 122.153°C and may get stuck on any of the two values for a few consecutive epochs. Unlike outliers, the noise faults in temperature readings are not correlated with noises in the humidity readings.
**Detection.** A primary identifier of a noise fault is the variance. For example, if the variance of reported sensor data is entirely different from the expected variance as given by the sensor data sheet, then it is a definitive fault. The data identified this way is invaluable and cannot be discarded once marked as a natural resemblance of the true phenomenon.

**Calibration Errors**

**Characteristics.** Calibration is becoming a highly prioritized requirement for every sensor network deployment [44, 50, 82, 106, 122]. Calibration is usually performed before, during, and after deployment to achieve the highest accuracy of the collected sensor data. Depending on the type of sensors, sometimes pre-deployment calibration is adequate. Nevertheless, some sensors need to be repeatedly calibrated over the course of deployment due to the change in their calibration equation and exposure to the environment. Usually higher cost sensors require less calibration and hence experience less calibration faults. One example of sensors that need to be continually calibrated is the ISE (Ion-selective Electrodes) sensor used in the Bangladesh deployment [106]. Calibration is the mapping from the actual sensor readings to the expected readings resembling the true phenomenon [106]. The mapping function contains two calibration parameters: (i) the bias or gain which relates to the value needed to drive the expected rate of change of the sensor readings; and (ii) the offset which is a constant distance from the true phenomena. The acceptable values of these two parameters are usually given in the sensor data sheet. Both the bias and the offset parameters can change over time given the dynamics of the environment and the manufacturing cost of the sensor. If the actual readings become distant from the true physical phenomena, then we say that sensor data has encountered an offset fault. On the other
hand, if the readings bear variances higher than the expected rate of change, a gain fault can be observed within the data. Finally, if both parameters change, this change manifests itself as a drift fault. The three calibration faults were elaborately discussed in [88].

Detection. Apparently, it is difficult to detect calibration errors in most cases due to the difficulty of distinguishing between a potential calibration fault and a true measurement of the phenomenon. Apart from this, spatial correlation may be leveraged to identify mis-calibrated sensors. Nonetheless, sensor readings from mis-calibrated sensors should not be discarded as an appropriate formula can be applied off-line to correct the faults.

Out of Range / Clipping Faults

Characteristics. Sometimes, sensor readings may lie outside the sensitivity range of the transducer or exceed the maximum value allowed by the Analog to Digital converter (ADC). The latter is a manifestation of a clipping fault. An example of an out-of-range fault was reported in the Bangladesh deployment [106] while measuring the concentration of chloride. There were cases when the concentration measurements were high enough that could not be captured by the sensitivity range of the transducer. In [106], the sensitivity range is referred to as the total detection range of the calibration curve and any sensors that report readings outside this range are considered BROKEN. As for clipping, although not necessarily a fault, it still implies an expected anomalous behavior. In the IBRL deployment, a clipping fault or anomaly is spotted when very high light intensities are clipped at the peak. True light values may even reach 100,000 Lux which corresponds to full sunlight. However, the light sensor in the IBRL deployment could only measure up to 1847.36 Lux (see Figure 3.7). Despite the fact that there are light readings very close to the lower ADC
Figure 3.7: Clipping faults exhibited by node 20 in the IBRL deployment. The light readings clipped at the highest peak are outside the range of possible ADC values.

bound (zero), we still cannot conclude that these are indeed clipping faults [88]. Ideally, clipping faults are similar to “stuck at” faults. The difference between the two is that, with clipping, the constant behavior is observed only at the peaks of the possible ADC range.

**Detection.** Clipping faults are indeed similar to “stuck at” faults and hence all detection mechanisms used for “stuck at” faults can be applied to identify clipping faults. The only difference between the two faults is that, with clipping, the constant behavior is observed only at the peaks of the data range. Leveraging spatial correlations, we can further identify a clipping fault which should not be discarded as it resembles an expected anomalous behavior.

**Connection / Hardware Failures**

This and the next fault in the list are in fact primary root causes of other sensor data faults (such as spike, “stuck at”, and noise faults). We list them here for the sake of completeness.
and consistency with the earlier work of Ni et al. The primary cause of this fault is short circuits created due to direct contacts with the environment (e.g., water contact, mud, etc. [50, 106, 122]). As we discussed earlier in Section 2, this fault manifests itself as very high or very low readings and could also resemble an out-of-range fault. Other causes of hardware failures include sensor aging and loose wire connections. It is important to note that it is not always feasible to identify the root cause of an unknown hardware failure, but once a hardware fault is spotted, generated sensor readings during the period of fault should be discarded.

**Low Battery Levels**

Just like hardware failures, low battery is a major cause of other sensor data failures (e.g., spike fault followed by a “stuck at” fault [11], noise fault [106] in several sensor network deployments [11, 106, 122, 126]). In addition, as mentioned in Chapter 2, hardware failures can be highly correlated with low battery faults. A short circuit can cause an unusual drop in battery level [122]. Maintaining power profiles using for example tele-diagnostic power tracers [58] may help identifying low battery failures. More often, data generated during a low battery period are harmful to the decision making process and should be discarded.

Sensor data faults are very common in sensor network deployments. Uncovering the root cause of such faults is a task that is sometimes very difficult. **Software bugs** such as the driver bug of the wind sensor in the SensorScope deployment [50] and **unexpected conditions** such as unintentional covering of the entire sensor mote [8] are hard to track. The two conditions were discussed earlier in Chapter 2. In the case of an existing software bug in the sensor driver, spatial correlation among the sensor nodes does not help at all if the
Table 3.3: Measurement Faults, Their Root Causes, and Detection Features

<table>
<thead>
<tr>
<th>Type</th>
<th>Example Deployments</th>
<th>Root Causes</th>
<th>Main Detection Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>outlier (SHORT)</td>
<td>IBRL [11], SensorScope [50], Bangladesh [106], NIMS [55], CDI [122]</td>
<td>unknown</td>
<td>rate of change</td>
</tr>
<tr>
<td>Spike</td>
<td>IBRL, SensorScope, Bangladesh</td>
<td>- low battery</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- sensor failure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- short circuit</td>
<td></td>
</tr>
<tr>
<td>stack-at (CONSTANT)</td>
<td>IBRL, SensorScope, NAMOS [3]</td>
<td>- sensor ADC malfunction</td>
<td>variance</td>
</tr>
<tr>
<td>NOISE</td>
<td>IBRL, SensorScope</td>
<td>- low battery</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- hardware failure (short circuit)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- poor calibration</td>
<td></td>
</tr>
<tr>
<td>calibration error</td>
<td>Bangladesh</td>
<td>- change in calibration equation</td>
<td>spatial correlation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- exposure to environment</td>
<td></td>
</tr>
<tr>
<td>clipping / out of range</td>
<td>IBRL, Bangladesh</td>
<td>- reading outside the range of the ADC</td>
<td>variance accompanied by spatial correlation</td>
</tr>
</tbody>
</table>

3.2 Malicious Behaviors

The second type of anomalies prevalent in sensor networks is malicious behaviors (network and data). This anomaly is more frequent in sensor networks deployed in hostile environments [44]. Ideally, there are two types of adversaries that can introduce a malicious behavior in a sensor network: (i) insiders; and (ii) outsiders [56, 130]. The resource capability of outsiders can be unlimited; an outsider with a laptop-class device is capable of jamming the entire network using a long-range transmitter to emit signals that interfere
with the same channel frequencies used by the sensor network [130]. A laptop-class outsider can also eavesdrop on or capture every single message flowing into the targeted sensor network. Moreover, he/she can replay messages or inject new data into the network. Nevertheless, outsiders cannot access the code of a sensor node and cannot modify (encrypted) data generated by each node. In contrast, insiders have a special access to the sensor network by compromising one or more nodes eventually laying hands on the security keys, code, and data stored in these nodes. Compromising a sensor node usually requires very little or no effort [42]. Once a node is compromised, it becomes easy to launch attacks that may affect either the overall network behavior or the data generated by that node. It is further possible to use another more powerful laptop-class device to attack the network (i.e., generate a malicious behavior) after stealing all the information and identity of the compromised node.

Existing link layer encryption and authentication techniques [57, 93] can serve as preventive measures against outsider attacks. Most of the possible security attacks that are launched by an outsider can be prevented by denying access of an adversary to join the network using authentication with globally shared keys. However, these mechanisms are rendered useless in the face of an attack launched by a laptop-class insider or a compromised node. Even with encryption and authentication techniques in place, several anomalous behaviors due to an insider attack may still exist in a sensor network deployment. Intrusion detection systems (IDSes) [18, 128] take one step ahead of preventive measures in an attempt to detect such anomalies. An IDS is a system that monitors/audits host or network data for suspicious activities. Based on the audit data, an IDS for sensor networks should be able to detect both the malicious network behaviors that influence the overall performance
and the forged data that hinder the fidelity of sensor data streams. There are two types of
IDSes: (i) misused or signature-based; and (ii) anomaly-based. In the former, a database
of attack signatures is established a priori from the audit data. Any attack that matches a
signature in the database can be identified. On the other hand, anomaly detection based
IDSes typically mark those behaviors that deviate from the normal behaviors as anomalies
(attacks).

In the following discussion, we present the major attacks against sensor networks that
either affect the data or introduce a malicious network behavior. These attacks can be
generated by either an outsider or an insider. However as mentioned above, a simple au-
thentication and encryption technique is adequate to stop most outsider attacks (e.g., pas-
sive eavesdropping, sinkhole, Sybil, selective forwarding). Therefore, we mainly focus on
compromise behaviors or active attacks launched by a compromised node or laptop-class
insider.

### 3.2.1 Malicious Data Behaviors

**Attack Characteristics.** When an adversary captures the sensor node, he/she can re-
program it to inject falsified readings into the network. If a significant number of nodes is
compromised, data forged by these nodes will highly impact the decision making process
at the base station. An adversary can also alter readings sent by neighboring nodes before
forwarding the data messages towards the sink. This, in turn, compromises the integrity of
the sensor data. If a sensor network employs in-network processing or aggregation \(^{77,78}\),
false data can further mislead the aggregated results computed by the aggregation points
including the base station.
Compromise behaviors injected in sensor networks can be classified as either (i) naive or (ii) smart behaviors. Naive compromised nodes always send falsified readings, whereas smart compromised nodes inject falsified readings in an intermittent manner along with legitimate data. In both scenarios, a compromised node may send highly deviated readings or readings which are closer to the normal behavior of other sensor nodes, varying the amount of harm to the fidelity of the data. Figure 3.8 shows both naive and smart compromise behaviors with different degrees of deviations exhibited by three nodes (nodes 12, 17, and 20) during two hours of our lab deployment.

During the process of data falsification, an adversary may also decide to change the behavior of the data to be identical to any of the natural sensor data faults discussed in Section 3.1.3. Sometimes fault models themselves can be close in characteristics to a naive
or smart compromise behavior. An example of this behavior is the accelerometer readings in [112]. In this case, it is very hard to distinguish between a fault and a malicious attack. In addition, an adversary can manipulate its data or its neighbors’ data in a manner that is much harder to detect.

Detection. Just like with measurement faults, there are several proposed mechanisms such as reputation-based mechanisms [36] that detect malicious data behaviors caused by a compromised node. In general, detection is carried out by using statistical techniques and leveraging temporal correlations as well as spatial correlations among neighboring nodes. Rather than addressing complete compromise of a sensor network, all these techniques hope that less than half of the neighboring nodes are compromised and the detection accuracy degrades with the rates of false positives and false negatives increasing no faster than a rate proportional to the ratio of compromised nodes to the total number of nodes in the network [56].

3.2.2 Malicious Network Behaviors

Unlike malicious data behaviors, malicious network behaviors do not directly influence the fidelity of the sensor data, rather, they impact the overall network performance. The most common security attacks launched by a compromised node or a laptop-class insider which introduce malicious behaviors into the network are [130,137]: selective forwarding attacks; sinkholes; wormholes; Sybil attacks; spoofing and replay attacks; and Denial of Service (DoS). A significant number of proposed IDSes [18,76,141] and compromise detection mechanisms can identify malicious network behaviors (such as rule-based, machine-learning-based, etc.). IDSes offer much higher detection generality than tech-
niques targeting a specific spectrum of attacks. A typical IDS may run on the base station, at every node in the network, or at selected nodes (called monitor nodes). For a survey of proposed IDSes in sensor networks, the reader is referred to [30]. In this section, we describe each attack separately and discuss potential ways to detect it.

Selective Forwarding

**Attack Characteristics.** Once a sensor node is compromised, it becomes trivial for the adversary to selectively forward flowing traffic. This means that a compromised node may drop a portion of the packets (control and data) passing through it. Adversaries may either choose to drop all packets of the same type or deny packets from a number of source nodes. If every packet is being dropped, the compromised node is said to form a *blackhole*.

**Detection.** A selective forwarding attack (or a blackhole) can be detected by a more generic IDS or by a detection mechanism specifically designed for this type of attack (i.e., assuming no other attacks will be launched by the adversary). An example of the latter is the multi-hop acknowledgment based scheme proposed by Yu and Xiao [140]. In this scheme, a number of selected intermediate nodes called ACK nodes over the path from the source to the base station are responsible for notifying the source of any suspicious drop. Upon detecting an event, the source node sends a report packet containing a pre-defined ACK counter (which is decremented by one at every hop) towards the base station. The report packet propagates downstream by every hop and only ACK nodes reply with an acknowledgment. If an intermediate node does not hear a specific number of ACKs after some time, it suspects its downstream neighbor and sends an alarm packet upstream notifying the source node of this suspicious activity. After collecting a number of alarm packets,
a source node filters out false alarms and identifies the malicious nodes where the dropping might have taken place. A downstream detection mechanism was also proposed where the base station can reveal packet losses based on the range of lost sequence numbers communicated by intermediate nodes downstream. Once malicious nodes are identified, both the source and the base station can notify the routing layer to exclude such nodes from the forwarding path.

IDSes follow a different approach. The IDS proposed in [18], for instance, runs on monitor nodes. A monitor node listens to neighboring traffic and applies a set of rules to identify misbehaviors. If the number of failures is higher than a certain threshold (i.e., expected amount of occasional failures), a flag is raised. The “retransmission rule” is used to detect selective forwarding attacks (or blackholes). This rule states that if the monitor node, in its promiscuous mode, did not receive the same packet intended for its next hop, a failure counter is incremented.

Sinkhole

**Attack Characteristics.** This is one of the most severe routing layer attacks in sensor networks [56, 65]. Figure 3.9(a) illustrates a typical form of a sinkhole attack. The compromised node is equipped with a long range transceiver that can directly reach the base station. The adversary launches the attack by advertising this link to the neighbors. Inherent to most of the underlying routing protocols such as CTP, the neighboring nodes choose the compromised node as their next hop and re-advertise their updated routing information (e.g., path quality). In turn, nodes distant from the compromised node may elect the advertised path and update their routing information accordingly. This eventually enables
the compromised node to suck the entire network traffic in a certain region towards itself.

The adversary can also use a wormhole as discussed later to launch a sinkhole attack (see Figure 3.9(b)). Once a sinkhole is created, it becomes trivial to launch other attacks such as the selective forwarding attack discussed earlier.

**Detection.** Ngai et al. [87] proposed an intrusion detection algorithm that defends against a sinkhole attack. In their work, the base station is assumed to have the location information of all nodes in the network. The algorithm first identifies a number of suspected nodes. Suspected nodes are those that report data which are inconsistent with the other nodes in the network. The algorithm then forms a circle covering all suspected nodes. All nodes inside the circle are called the affected nodes. Once the list of affected nodes is formed by the base station, the base station forms a request message containing a timestamp, the list of all affected nodes, and the base station’s private key and floods it into the network. The timestamp and the key are used to avoid replay attacks. Once an affected node receives the request, it checks to see if its id is in the list. If so, it sends a reply message, over the reverse path, to the base station containing its own id, the id of the next
hop, and the path cost (e.g., link quality of the path from the node to the base station). Such information indicates the network flow (routing patterns) and is maintained in the node’s routing table. Upon receiving all replies, the base station constructs a tree using the next hop information and discovers the sink hole represented by the root of the tree since all information flow towards the root node. Moreover, an algorithm is proposed to identify the sinkhole in the presence of a few compromised nodes that attempt to trick the base station into believing that a good node is the sinkhole hiding the identity of the real sinkhole. This algorithm exploits encryption and authentication and uses redundant paths to send the reply messages to the base station to avoid selective forwarding attacks initiated by the malicious nodes. It also attempts to identify the malicious nodes by exploiting inconsistencies within the collected network flow information at the base station.

**Wormhole**

**Attack Characteristics.** The wormhole attack was first introduced by Hu et al. [48] for ad-hoc networks. In this attack, a tunnel is typically established between two distant malicious nodes (e.g., compromised nodes in Figure 3.9(b)). Each malicious node uses a higher-than-usual transmission power to relay the network traffic flowing into it without modifying the contents of the forwarded packets. This forces nodes at two opposite parts of the network to believe they are neighbors. In a similar attack called the “Hello flood” attack, a form of a Denial of Service, the adversary attempts to convince every node in the entire network that it is a neighbor of the compromised node by broadcasting beacon messages using an extremely high transmission power. As a result, all messages generated by nodes that are at least two hops away from the malicious node will be lost. Similar to a
sinkhole, a wormhole can be used to launch other attacks in the network such as a sinkhole (Figure 3.9(b)), a selective forwarding attack, or a Sybil attack discussed later.

**Detection.** An RSSI-based mechanism was proposed in [95] which detects wormholes (and Hello floods) in the presence of no other attacks in the network. This scheme assumes that every node is aware of its geographical location and the sensor network is symmetric, static, and homogeneous. Upon reception of a message, each node computes the expected signal strength of the received signal using both the predefined transmission power and the distance between itself and the node that transmitted the message. If the difference between this value and the actual RSSI value reported by the radio chip is greater than a certain threshold, the received message is considered suspicious and the node transmitting the message is labeled as “suspicious.” The receiving node then broadcasts its findings to its neighbors who, in turn, reply with their opinions about the same suspicious node. The receiving node uses these opinions (votes) to determine if the suspicious node is launching a wormhole (or a Hello flood) attack.

The rule-based IDS in [18] uses the “radio transmission range rule” to detect a wormhole (or a Hello flood). This rule states that “all messages received by the monitor node in a promiscuous mode must be originated from one of its neighbors.”

**Spoofing / Replay**

**Attack Characteristics.** An adversary can change the compromised node’s routing information, spoof its own identity, or fabricate snooped link-layer acknowledgments. As for acknowledgment spoofing, most link-layer protocols used in today’s sensor network deployments use implicit and/or explicit acknowledgments. By fabricating an acknowledg-
edgment upon listening to neighborhood traffic, an adversary may be able to persuade the sending node to think that the destination node is alive while in fact it is being disabled by the adversary or has already failed. The adversary can also convince the sending node that a weak link is strong leading to selective forwarding since the link cannot deliver the packet. On the other hand, modifying/replaying routing information or impersonating another node’s identity may result in network failures such as routing loops and network partitioning. With identity spoofing in sensor networks, since a node can hear its neighborhood traffic, it can simply use one of its neighbors identities as its own to deteriorate the network functionality. For instance, an adversary (i.e., a compromised node) can spoof routing updates in CTP \cite{37} in order to create a routing loop between two nodes A and B. The adversary first picks up node A’s identity by listening on neighboring traffic. It then sends a fake routing update message to node B containing node A’s address as the source address. Node B then marks node A as its parent and rebroadcasts the update. In turn, node A snoops (overhears) node B’s update and marks node B as its parent \cite{56}. This causes traffic to circulate between both nodes endlessly until the loop is detected. The same methods described in Section \ref{3.1.1} can be used to detect such a failure. However, there will be no clear distinction between a natural network failure or a failure caused by spoofing.

Detection. Chen et al \cite{16} have proposed a scheme that exploits the spatial correlation property of the received signal strength (RSS) to detect and localize identity spoofing and thus to avoid routing loops. In this scheme, the RSS readings of every identity are clustered in a signal space using the K-means clustering algorithm. The RSS readings from the same physical location should belong to one cluster. If a malicious node claimed the identity of another node, its RSS readings would belong to a different cluster since it is distant from
the other normal node. Based on the distance between the centroids of the two clusters, the scheme can tell that the network is under a spoofing attack. The accuracy of this detection scheme is improved if the compromised node is located further away from the node with the stolen identity.

**Sybil Attack**

**Attack Characteristics.** Sybil attacks were first discussed in the context of peer-to-peer networks [27]. It is an example of an identity-based attack where the compromised node claims multiple identities. A taxonomy of the different forms of a Sybil attack on sensor networks was established in [86]. The authors distinguished between Sybil attacks using direct vs. indirect communication, stolen vs. fabricated identities, and simultaneous vs. non-simultaneous use of identities.

**Detection.** The authors in [86] proposed five techniques to verify the identity of a Sybil node (i.e., an imaginary node with a stolen or fabricated identity). These are: (i) radio resource verification; (ii) cryptographic key validation; (iii) identity registration; (iv) position verification; and (v) code attestation. Each one of these techniques may miss one or two types of the classified attacks except code attestation which is always capable of detecting any form of a Sybil attack.

**Denial of Service**

**Attack Characteristics.** A Denial of Service (DoS) attack eliminates a network’s capability to perform its expected functionality. All previous attacks discussed so far are mainly routing layer attacks. DoS attacks, however, can affect any of the five layers (physical, link, routing, transport, and application layer). Wood and Stankovic [134] constituted
a comprehensive survey of DoS attacks. In their survey, they have shown that most of these attacks can be countered using cryptography, authentication, or fairly simple preventive measures except for jamming. Jamming is a physical layer attack whereby a laptop-class insider can use a long-range transceiver to interfere with the radio frequencies used by the legitimate sensor nodes [130]. This may shut the entire network down in a short period of time. This can also happen if a small number of compromised nodes with similar radio capabilities as legitimate nodes that are randomly distributed across the network launch a collaborative jamming attack [130]. In addition, the Hello flood discussed earlier is another example of a DoS attack. It uses higher power to transmit “Hello” messages to the entire network. Finally, notice that network and node failures may also cause a DoS attack since such failures may render parts of the network non-functional.

Detection. The work in [83] surveys a number of detection techniques proposed for jamming. Typically, these techniques use signal strength, location information, or an estimate of the transmitter’s interference strength as the detection measure. As for Hello floods, techniques used to detect wormholes can similarly be used to reveal Hello floods as well. Finally, the rule-based IDS in [18] uses the “repetition rule” to detect a generic DoS attack. The repetition rule states that “the same message can be retransmitted by the same neighbor only a limited number of time.”

In summary, there are instances where proactive measures may be adequate to prevent malicious behaviors. Nevertheless, a total compromise of a sensor node or a small number of nodes can break these measures. Rapidly detecting a malicious behavior when the adversary launches the attack is another line of defense. Once the malicious behavior is identified, further actions can be taken to eliminate the compromised node. Table 3.4 sum-
Table 3.4: Malicious Network Behaviors, Attack Characteristics, and Possible Detection Approaches

<table>
<thead>
<tr>
<th>Type</th>
<th>Attack Characteristics</th>
<th>Potential Existing Detection Mechanisms</th>
</tr>
</thead>
</table>
| selective forwarding | - an adversary (i.e., compromised node) may drop some or all packets (control and data) passing through it.  
- if all packets are being dropped, the node becomes a blackhole | - rule-based IDS [18]  
- multi-hop ack based scheme [140] |
| sinkhole        | - adversary possesses a long range transceiver that can reach the base station  
- adversary sucks all traffic in certain region towards itself by advertising its high quality link | - intrusion detection algorithm proposed by Ngai et al [87] |
| wormhole        | - adversary establishes a tunnel between two distant compromised nodes using a higher-than-usual transmission power  
- nodes at opposite sides of the network think they are neighbors resulting in packet losses  
- if all nodes become neighbors of the compromised node, this leads to a Hello flood | - rule-based IDS  
- RSSI-based mechanism [95] |
| spoofing / replay | - change compromised node’s routing information  
- spoof node’s own identity  
- fabricate snooped link-layer acknowledgments | - data path validation in CTP  
- network diagnostic tools (e.g. AD)  
- exploiting spatial correlation property of RSS [16] |
| Sybil attack    | - compromised node claims multiple identities | - radio resource verification, cryptographic key validation, identity registration, position verification, and code attestation [86] |
| Denial of Service | - eliminate network’s capability to perform its expected functionality  
- shut down the entire network in a short period by jamming with a long-range transceiver  
- hello floods | - rule-based IDS  
- RSSI-based mechanism [95]  
- techniques relying on signal strength, location estimation, or transmitter’s interference strength [83] |

3.3 Events

The last type of anomalies in sensor network deployments is *events*. In event-driven application deployments such as VigilNet, the application administrators are interested in these abnormal instances that deviate from the normal setting. They would like the deployed network to notify them of any inconsistent behavior within the monitored area. Figure 3.10 illustrates few examples of potential events which occurred in the IBRL deployment. Usu-
(a) Node 37 shows higher temperature values due to its physical placement close to the kitchen and node 14 is believed to exhibit deviated behavior at the end due to unusual interference [104].

(b) The first four hours starting midnight March 1. During short periods, nodes 10 and 16 were potentially exposed to regions of higher heat than the other regions where the rest of the nodes are deployed.

Figure 3.10: Potential events prevalent in the IBRL deployment.

ally, events manifest themselves as sudden abnormal jumps in the data. However, the amount of jump and the behavior of the data after this differs from one situation to another. Due to the high uncertainty involved in predicting the behavior of an event, it is important to study the types of events desired to be detected by the deployed sensornet. Often, the characteristics of the event are known a priori or by experimentations [132].

**Detection.** Three approaches are common for event detection: (i) fully centralized; (ii) loosely distributed; and (iii) fully distributed. In the centralized approach, all data points are first gathered at the base station and then a detection algorithm is applied over the collected data. This incurs high detection delay and high energy cost. In the second approach, i.e., loosely decentralized, events are tracked at each sensor node individually. When an event is detected, it is reported to the base station for further verification with other nodes’ suggestions. Finally, the fully decentralized approach requires the sensor nodes to exchange information in order to detect a potential event in their vicinity without direct help from the base station. Wittenburg et al [132] proposed a fully decentralized scheme...
for event detection and qualitatively compared it with other schemes. Their scheme is proven to be superior to others as it consumes less processing and memory resources, does not require any cluster-like infrastructure, leverages pre-training to improve accuracy, and allows sensor nodes to collaborate.

A key factor in detecting events is to be able to disambiguate between a faulty measurement and an event that has happened in a specific region [39]. Fault-event disambiguation is not a trivial task. However, if spatial correlation is exploited, geographically independent faults can be distinguished from events which are based on correlated readings observed by a number of sensors in a region or event boundary. The work of [25] proposed a spatial data mining based mechanism to detect event boundaries in presence of faulty measurements.

**Data-Centric Anomalies.** Having classified anomalies into three major categories (natural faults, malicious behaviors, events), we now identify these anomalies that only influence the collected data samples (data-centric). Measurement faults, malicious data behaviors, and events are all examples of a *data-centric* anomaly. Data faults and malicious data behaviors hinder the quality, integrity, and/or trustworthiness of the sensor data whereas events manifest as abnormal data. Some detection mechanisms focus on data-centric anomalies in general and do not necessarily distinguish a measurement fault from an event from a malicious data behavior. For a survey of such mechanisms, the reader is referred to [143].

### 3.4 Characteristics of Anomaly Detection Algorithms

After establishing a comprehensive taxonomy of the different types of anomalies in sensor network deployments, we now discuss the challenges in designing an anomaly detection
mechanism or algorithm. These challenges are common to all types of anomalies discussed in this chapter. An anomaly detection algorithm should be:

- **Resource-aware**: The most prevalent resource constraint in the majority of sensor network deployments is “energy.” Communication usually accounts for the highest amount of energy consumption. Processing power can also be high if advanced and more complex operations are performed at the sensor node (e.g., surveillance [44]). Memory, on the other hand, may not be a constraining resource in recent days [127]. However, writing to flash memory is still very expensive in terms of power consumption. Given these constraints, an anomaly detection algorithm should try to eliminate any additional burdens on the battery capacity and, if possible, utilize the remaining space in the data memory before attempting to write to much larger flash memory.

- **Simple**: The cost of computation may not be the major concern in sensor network deployments. However, algorithms that run on low-power resource-constrained embedded devices such as sensor motes should not incur a high computational complexity. Complex algorithms can incur delay and may unexpectedly crash the operating system running on top of the sensor motes. Moreover, it is difficult to debug or track the operation of a large network of tiny sensor motes to ensure the correctness of the detection algorithm before deployment. Hence, simplicity of algorithms may ease the debugging process.

- **Accurate**: As any other detection algorithm, an anomaly detection algorithm designed for sensor networks should achieve a high detection accuracy. Anomaly detection schemes that yield high rates of false positives and false negatives are not desirable.
• Rapid: Detection latency is an important measure of the effectiveness of an anomaly detection algorithm. A low detection latency is desirable in all cases. In an event-driven application, for example, events should be identified in a timely manner. Natural faults and malicious behaviors should also be revealed before the entire network goes down or the sensor data is severely contaminated with inconsistent data points.

• Scalable: Most deployments discussed in Chapter 2 constituted a descent number of sensor nodes. A recent deployment of GreenOrbs [114] consisted of 349 nodes and is still in expansion. Therefore, detection algorithms that run perfectly in a lab environment with few sensor nodes may not necessarily adapt to networks at large scales.

• Adaptable: WSN deployments usually involve incremental installation of new nodes in the field over time. The detection algorithm should tolerate any addition of new nodes or removal of dead nodes.

• Robust: Although we have put down a comprehensive taxonomy of anomalies, sometimes it is difficult to anticipate the true behavior of an anomaly (e.g., a new attack on sensor networks). Therefore, it is highly desirable that a detection algorithm identifies unknown anomalies such as novel events, unexplored future faults, or newly emerged malicious activities.

• Secure: This challenge is perhaps more relevant to sensor networks deployed in hostile environments. An algorithm designed for revealing a malicious behavior should itself be resilient to attacks.

• Practical: In this chapter, we discussed a number of detection and mitigation techniques
against anomalies in sensor networks. However, most existing techniques have not been tested in real-world deployments. This calls for conducting real-world experiments to prove the effectiveness of such techniques under realistic environments. Moreover, in recent sensor network deployments for monitoring applications, several other services are employed such as sensor data collection, parameter dissemination, etc. Rather than interrupting the normal operation of the sensor network, an anomaly detection algorithm or service should run alongside existing protocols.

3.5 Related Work

An earlier attempt was made by Jurdak et al [54] to classify anomalies in sensor networks. However, the authors only addressed natural faults and did not elaborate on measurement faults. A measurement fault, as one type of a natural fault, is a very significant type of anomaly and was extensively described in our work for further classification following the same terminologies used in [88] and [112]. Malicious behaviors and events can also be considered as anomalies in sensor network deployments and hence are an essential part in our taxonomy.

Ni et al [88] compiled a study of sensor data faults prevalent in sensor network deployments and presented helpful features that can be used in the detection process. Both [137] and [130] surveyed the different types of malicious network behaviors and discussed a number of countermeasures. Our work borrowed the terminologies presented in these works to maintain consistency. However, we deal with anomalies in general rather than single measurement faults or malicious network behaviors. This work is more general in scope by providing an insight to the reader about “anything” that can go wrong in a sensor network
whether deployed in a friendly or a hostile environment by constructing a comprehensive taxonomy of such anomalies.

A number of surveys [30,108,136,143] exist in the literature that address the detection of a specific scope of anomalies discussed in this chapter. In [108] the authors compared among the various diagnosis and debugging tools which handle functional faults (network and node failures). The work of [143] surveyed the different mechanisms for detecting data-centric anomalies. Intrusion detection systems for identifying malicious network behaviors were discussed in [30]. Finally, Xie et al [136] studied a number of detection mechanisms for data-centric anomalies as well as malicious network behaviors. Our work in this chapter differs from these surveys in two aspects. First, the main focus of this chapter is the analysis rather than the detection of different types of anomalies that are prevalent in recent sensor network deployments. Second, the discussion in these surveys does not necessarily show how each detection mechanism deal with every individual anomaly. In contrast, our work gives the reader an insight on how to do so by presenting some existing solutions.

### 3.6 Summary

In this chapter, we established a taxonomy of different types of anomalies prevalent in sensor network deployments. We discussed each anomalous behavior and gave insights on how to potentially identify such a behavior using existing detection mechanisms. Finally, having advocated for anomaly detection mechanisms, we presented the challenges of designing a new mechanism suitable for real-world sensor network deployments. This comprehensive study can also help researchers narrow their focus to a specific type of anomaly or choose to implement a more generalized anomaly detection mechanism.
Chapter 4

Data-Centric Anomaly Models

The performance of anomaly detection algorithms is usually measured using the total residual error. This error metric is calculated by comparing the labels assigned by the detection algorithm against a reference ground truth. Obtaining a highly expressive ground truth is by itself a challenging task, if not infeasible. Often, a dataset is manually labeled by domain experts. However, manual labeling is error prone. In real-world sensor network deployments, it becomes even more difficult to label a sensor dataset due to the large amount of samples, the complexity of visualizing the data, and the uncertainty in the existence of anomalies. In this chapter, we propose an automated technique which uses highly representative anomaly models for labeling. This technique will be extensively used in consecutive chapters. We demonstrate the effectiveness of this technique in the next chapter through evaluating a proposed classification algorithm using the designed anomaly models as ground truth. We will show that the classification accuracy is similar to that when using manually labeled real-world data points.
4.1 Overview and Motivations

Evaluating the accuracy of anomaly detection algorithms requires matching the predicted labels—or clusters in case of an unsupervised algorithm—against a solid ground truth. Obtaining a highly accurate ground truth is by itself rarely possible. Often, algorithm designers simply rely on visual inspection to label anomalies and normal behaviors in the original dataset. Although this mechanism may achieve a good point of reference, it becomes tedious to inspect the entire data obtained from real-world deployments. In addition, inspectors are sometimes uncertain if a suspicious reading is in fact an anomaly unless they are experts in their domain. Although uncommon, algorithm designers may use metrics such as distance, density, or a running average to label anomalies in the original data \cite{142}. However, these labeling techniques are themselves anomaly detection mechanisms that have their own deficiencies.

An alternative approach to obtaining a ground truth is to inject artificial anomalies with known labels into the existing sensor dataset. Although this may entirely eliminate errors associated with mislabeling the original data, injected anomalies should follow models highly expressive of the true anomalous behaviors. To the best of our knowledge, this work is the first to propose such models. In \cite{112} and \cite{15}, most anomalies are injected using random noise generation with adjustable parameters. An injected outlier, on the other hand, is represented as an extremely large or extremely low reading. Outliers can also follow an injection model with specific parameters that control their values. Anomalies injected using random noise or outlier generation do not necessarily mimic the true behaviors observed in real-world deployments and hence a new injection method is required. Different sensor-
net deployments may exhibit different anomalous data behaviors depending on the type of sensing modalities being monitored and the sensor manufacturer. Existing sensor networks that are deployed for environmental monitoring purposes, mostly share similar characteristics. Sensors in these deployments typically measure a certain number of phenomena including temperature, humidity, and light intensity. To design our anomaly models, we choose to study the Intel Berkeley Research Lab (IBRL) deployment. Despite the fact that the IBRL deployment does not cover all possible anomalous behaviors, it still gives us a good basis for designing such models. One can extend these models to include new behaviors that may be seen in other deployments. In cases, some anomaly models (e.g., outlier models) may only require an addition to or modification of a list of values (e.g., true outliers), whereas in other cases, an entirely new model should be designed.

In addition to measurement faults, we also design models for injecting naive and smart compromise data behaviors. It is safe to assume that those two models are highly representative of the adversary’s intention to disguise himself/herself, while injecting falsified readings that hinder the quality and trustworthiness of the collected sensor data. Finally, events in event-driven sensornet applications often exhibit distinct characteristics. Given this fact, it is infeasible to design a unique model for generating all kinds of artificial events. As a result, application administrators may simulate possible events to be detected and based on the data behaviors, suitable models can be designed. The models designed in this chapter are based on a time-series analysis. This choice was made due to the fact that anomalies may also be injected into existing sensor network deployments in real-time. This enables online detection mechanisms, such as the one we propose in chapter 6, to rapidly evaluate their accuracy on the fly.
4.2 Modeling Anomalies

In the previous chapter, we analyzed the characteristics of data anomalies that can exist in real world deployments. Based on these characteristics, we now define mathematical models that are highly expressive of data-centric anomalies. Since sensor readings in real-world sensornet deployments arrive at the base station in real-time, we can define time-series models which act upon streams of sensor data that express a number of sensing modalities (e.g., a stream of temperature readings). To this end, a newly arrived reading can be mapped using the following general model:

\[
y_{p,k} = \begin{cases} 
\phi & \text{if } n \leq p < n + w \\
x_{p,k} & \text{otherwise}
\end{cases}
\]  

(4.1)

where \( p \) represents the current epoch, \( k \) is the sensing modality and \( y_{p,k} \) is the newly mapped sensor reading. This general model states that if the current epoch lies within a certain time window \( w \) starting at epoch \( n \), then we apply a specific mapping function \( \phi \) over the current sensor reading \( x_{p,k} \). The mapping function \( \phi \) is the function upon which the anomalous reading is generated and hence is dependent on the anomaly type. Here, it is assumed that current readings represent the true phenomena. If we are dealing with an existing sensor dataset such as the IBRL dataset, we would first filter out anomalous readings within a larger time window before applying the mapping, as we will do by filtering out outliers in the coming chapter. On the other hand, if sensor readings are being delivered in real-time, they may themselves be anomalous at the time of modeling, introducing additional noise that may affect the model accuracy. Therefore, anomaly modeling is usually preferred during the initial deployment when the nodes are presumably healthy.
Table 4.1: A Set of All Outliers In The IBRL Dataset Prior To Battery Voltage Drops Under 2.40 Volts

<table>
<thead>
<tr>
<th>Node 2</th>
<th>Node 4</th>
<th>Node 8</th>
<th>Node 20</th>
<th>Node 26</th>
<th>Node 32</th>
<th>Node 39</th>
<th>Node 43</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage (Volts)</td>
<td>0.0180369</td>
<td>2.14412</td>
<td>0.0346855</td>
<td>2.4341</td>
<td>2.42416</td>
<td>2.17553</td>
<td>18.56</td>
</tr>
<tr>
<td>Humidity (RH%)</td>
<td>-1854.52</td>
<td>-8983.13</td>
<td>-38.4</td>
<td>-8983.13</td>
<td>-4</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td>Light (Lux)</td>
<td>158.24</td>
<td>235.52</td>
<td>0</td>
<td>890.56</td>
<td>625.6</td>
<td>566.72</td>
<td>1847.36</td>
</tr>
<tr>
<td>Date</td>
<td>02-28</td>
<td>03-16</td>
<td>03-06</td>
<td>02-28</td>
<td>02-28</td>
<td>02-28</td>
<td>03-20</td>
</tr>
<tr>
<td>Epoch (Sequence Number)</td>
<td>849</td>
<td>50772</td>
<td>22835</td>
<td>1694</td>
<td>1773</td>
<td>1834</td>
<td>61690</td>
</tr>
</tbody>
</table>

4.2.1 Outliers

In the IBRL deployment, a few nodes exhibited one or a couple of outliers prior to the time where their battery voltage began to drop under the normal operating voltage of the temperature and humidity sensors. Table 4.1 lists all these instances of outliers. Outliers would appear in all sensing modalities (temperature, humidity, and light) concurrently. The outliers in the table show no correlation whatsoever with the sampled reading in the previous epoch or, in other words, the expected sensor reading at the time of outlier. We have also seen that the occurrence of outliers correlates well with the occurrence of outliers in the voltage values. However, we cannot model outliers as a function of voltage since there is no clear dependency between the two values (sensor reading outlier and voltage outlier). Therefore, outliers can be modeled as $\phi = c_k$ where $c_k \in C_k$ is a specific outlier to be injected at epoch $n$. Notice that the time window $w$ in Eq. 4.1 is equal to one since an outlier only happens at a single epoch. Several outliers can be injected at different epochs as this was the case with nodes 20, 32, and 39 in Table 4.1. The set $C_k$ holds all observable outliers for sensing modality $k$. Referring to Table 4.1, the set $\{-38.4, 0.016, 3.4068, 90.9208, 175.681, 385.568\}$ is mapped from SI units to the raw ADC values to form the set $C$ of temperature outliers in the IBRL dataset. This set can be
expanded to cover more outlier instances. An outlier instance can also be randomly generated as either a much lower value or much higher value than the possible values of the true phenomenon [112]. However, a randomly generated outlier may not be as representative as known outliers in existing deployments.

4.2.2 Spike Faults

As discussed earlier, the IBRL dataset contains a large number of spike faults within humidity and temperature readings. To model spike faults, the model should highly approximate the spike behavior once it takes place. As the relationship between spike data points and epochs is non-linear, a high-degree (i.e., \( > 4 \)) polynomial regression can be applied to fit the faulty readings. The following \( r \)-th order polynomial equation can then be used to estimate the spike value of a sensing modality \( k \) at each epoch:

\[
\phi = \theta(\beta_{0,k} + \beta_{1,k}q_k + \beta_{2,k}q_k^2 + \ldots + \beta_{r,k}q_k^r)
\]  

(4.2)

\( \theta() \) is a rounding function since sensor readings are injected in ADC format. The response dependent variable in this case is \( \phi \) and the predictor independent variable is the current epoch increment \( q_k = (p_k - n) \). To determine the current epoch increment, the value of the epoch where the spike has just occurred (i.e., \( n \)) is subtracted from the current epoch. Finally, \( \beta_{0,k} \) represents the intercept, and \( \{\beta_{1,k}, \ldots, \beta_{r,k}\} \) are the polynomial coefficients of the regression equation. To inject a spike fault, one would choose \( n \) to be the very first incident when the sensor node began to exhibit a spike fault and \( w \) to be the spike period (e.g., \( w = 2910 \)). Recall that epochs represent sequence numbers and hence, for each sensor node, not all epochs are available since packet loss is certain during the spike
Figure 4.1: Various modeling techniques for spike faults exhibited by two sensor nodes in the IBRL deployment: nodes 6 and 18. For each node figure, the upper region plots the entire period of the spike fault followed by the steady behavior of a stuck-at fault. The lower region plots the first two hours of the spike period.

Although regression can be a good approximation of the original spike data, it does not model the spike behavior as accurate as numerical methods (e.g., linear interpolation). Hence, we further propose two interpolation techniques: (i) simple linear interpolation and (ii) spline interpolation. The resulting fitting curves for the three techniques are plotted in Figure 4.1. The figure also shows the original data points representing a spike fault exhibited by both node 6 and node 18. Data points were obtained by mapping the temperature values from degree Celsius to their corresponding ADC format. Apparently, the fitted curve obtained by simple linear interpolation and spline interpolation passes through all faulty data points. Nevertheless, missing packets will have to be interpolated as well. If the number of missing packets during the spike period is very large, some of these packets may be interpolated at the peaks of the fitted curve. This is the main drawback of using...
the spline interpolation method. The smoothed curve can be leveraged to inject exact spike data points if such points occur at the same epochs as the original spike points. Spline interpolation (or linear interpolation in general) requires all the dependent and independent variables to be stored in memory. If fault injection is to take place offline, this adds no burden. However, if faults need to be injected at the sensor level, sensors do not have the capacity required to store this huge amount of points. Hence, regression is preferred in the latter case as only the regression function needs to be stored in the node’s data memory and is used to generate spike points given the current epoch value.

Finally, ADC readings for the temperature and humidity sensors highly depend on the sensor manufacturer. For instance, Sensirion temperature and humidity sensors used in the IBRL deployment sample readings as 14-bit ADC output whereas other sensors such as the temperature sensor on the MTS300 sensor board can only sample 10-bit of ADC output.

4.2.3 Stuck-at Faults

In the IBRL dataset, most stuck-at faults maintain steadiness until the end of a node’s lifetime. This steadiness follows a spike fault. The sensor reading may also become stuck at a constant value which follows as a result of a noise fault. Since these two values are consistent across nodes, stuck-at faults can be modeled as \( \phi = c_k \) where \( c_k \in \{c_1, c_2\}_k \). For instance, for humidity, \( c \) is chosen to be the converted ADC value of either 114.894 or -3.91901 R%. Although a stuck-at fault does not always exhibit a zero variance over time (few faulty readings are slightly deviating from \( c_{1,k} \) or \( c_{2,k} \)), the only chosen two constant values are adequately representative of a stuck-at fault and these few deviating readings can be considered as part of the fault. As constant faults following a spike fault usually carry
out until the end of the deployment or until the node energy is entirely depleted, the time window \( w \) can be specified as large as a week to 10 days (see Figure 3.3(b)). Furthermore, \( n \) can carry on after the spike fault has ended \( (n = n_{\text{spike}} + w_{\text{spike}}) \).

### 4.2.4 Noise Faults

As pointed out in chapter 3, noise faults in the IBRL deployment may take place at the end of the node’s lifetime when the battery is being depleted. Similar to stuck-at faults, we model this behavior as \( \phi = c_k \) where \( c_k \in \{c_1, c_2\}_k \). However, this behavior is repeated over \( m \) times whereby in each iteration we switch between \( c_{1,k} \) and \( c_{2,k} \). Furthermore, in every iteration \( w \) is randomly chosen to be small enough in order not to lead to a constant fault otherwise. More formally, a noise fault exhibited by a sensing modality \( k \) over a number of windows \( W \) can be given as

\[
Y_{\text{noise},k} = \sum_{i=0}^{W} \left( \sum_{p=n_i}^{n_i+(w_i-s_i)} c_{1,k} + \sum_{p=n_i+(w_i-s_i)}^{n_i+w_i} c_{2,k} \right)
\tag{4.3}
\]

and \( n_i = n_{i-1} + w_{i-1} \) when \( i > 0 \). Other types of noise faults are possible in other deployments as mentioned in [88]. These faults may be modeled using a white Gaussian noise with zero mean and \( \sigma \) variance [113] in a way similar to a naive compromise behavior as we will discuss shortly.

### 4.2.5 Clipping Faults

Similar to stuck-at faults, clipping faults can be modeled by maintaining a constant value over a number of epochs (i.e., \( \phi = c_k \)). The only difference is that, with clipping, the constant value represents the maximum limit bounded by the number of bits of the analog
to digital converter (ADC). As mentioned in Chapter 3, the light sensors in the IBRL deployment could generate readings up to 1847.36 Lux. The minimum ADC value is omitted from the list of clipping model parameters.

Notice that the above fault models and their parameters are designed based on the IBRL dataset. If the sensornet is operating in a different context, model parameters have to be updated accordingly. Furthermore, additional models may have to be designed for other contexts. Still, the proposed real-time fault models provide a common ground for modeling faulty data in cyber-physical systems especially those used for environmental monitoring.

4.2.6 Events

In the previous chapter, we discussed the characteristics of potential events that could take place in a real-world scenario. As mentioned, it is infeasible to model all events using one single model and therefore event detection mechanisms consider few known events to be identified [132]. To this end, we merely consider one of the potential events exhibited by node 16 during the IBRL deployment (see Figure 3.10(b)). This behavior may be modeled using a random offset $o_k$ followed by a very small white Gaussian noise ($\phi = o_k + x_{p,k} \pm \sigma_{p,k}$). The node then recovers after a time window $w$ has passed. Other event models can be designed in a similar fashion.

4.2.7 Compromise Data Behaviors

Recall that compromise data behaviors can be either naive or smart. Both behaviors can be modeled as $\phi = x_{p,k} \pm \sigma_{p,k}$, where $\sigma_{p,k}$ is the variance of a white Gaussian noise with zero mean. In case of a naive behavior, the additive noise is added continuously over the entire injection window $w$. On the other hand, the additive noise is only added during half of the
current injection window if it is a smart compromise behavior. If the number of injection windows is $W$, then a smart behavior for a sensing modality $k$ is formally expressed as:

$$Y_{\text{smart},k} = \sum_{i=0}^{W} \left( \sum_{p=n_i}^{n_i+w_i} (x_{p,k} \pm \sigma_{p,k}) + \sum_{p=n_i+w_i}^{n_i+2w_i} x_{p,k} \right)$$  (4.4)

where $n_i = n_{i-1} + w_{i-1}, i > 0$. During the process of data falsification, an adversary may also inject anomaly models identical to the fault models discussed earlier. Moreover, in most sensor network deployments, a behavior of a noise fault highly matches that of naively injected falsified readings [112]. Therefore, such faults can follow the same model of a naive compromise behavior.

Finally, the amount of deviation of compromised readings overtime should be high enough to impose a threat (e.g., temperature variations outside the normal variations of the true temperature readings which happen due to calibration errors and other factors), otherwise, the final result would lie within the expected range.

### 4.3 Summary

In this chapter, we proposed models to generate sensor data anomalies which closely represent those prevalent in real-world sensor network deployments. A common ground truth can be established by injecting anomalies according to their models into an existing sensor dataset or deployment. Automatic labeling of the data eliminates any potential human errors of mislabeling some of the data samples through tedious visual inspections. In the coming chapter, we will show how expressive the designed models are by injecting a number of faults into an existing real-world sensor dataset. The resulting ground truth serves as
a reference to measuring the accuracy of a classification algorithm we propose. We achieve similar accuracy measures as those when using real faults for training the classifier.

In this chapter, we designed anomaly models based solely on the IBRL deployment and one of our lab deployments. The IBRL deployment exhibits most of the anomalous behaviors that can be found in an environmental monitoring application. However, sensor data faults still depend on the sensor type and manufacturer. In the IBRL deployment, only a certain number of sensors were used, namely temperature, humidity, and light sensors. Other types of sensors manufactured by other manufacturers may exhibit different faulty behaviors than those in the IBRL deployment as we saw in Chapter 3. We plan to come up with new expressive fault models for such data-centric anomalies. We also wish to model more interesting events such as unauthorized office access in the near future. Furthermore, other than data anomalies, we intend to establish a solid ground truth representing network and node anomalies as well. Network and node failures impact the network performance and lifetime, and thus detecting such anomalies is very desirable despite the fact that it is not the main objective of this thesis.
Chapter 5

Data-Centric Anomaly Detection in Sensor Networks

As we have seen from the previous chapters, data-centric anomalies are highly prevalent in sensor network deployments. To meet one of the main goals of this dissertation, that is of enhancing quality assurance of sensor data acquisition schemes, it is crucial to first identify data-centric anomalies. There have been a number of data-centric anomaly detection schemes proposed over the past decade [143]. These range from rule-based, to statistical-based, to graph-based, to machine learning based mechanisms. The latter have proven to be effective in detecting measurement faults. In the next section we discuss these schemes and pinpoint their weaknesses. In Section 5.2 we propose a new machine learning based data-centric anomaly detection framework which is capable of accurately identifying sensor data faults as well as compromised data behaviors. We summarize the chapter in Section 5.3

5.1 A Taxonomy of Data Mining Based Data-Centric Anomaly Detection Techniques

Data mining techniques for anomaly detection have been widely used over the years in various domains including intrusion detection, fraud detection, medical anomaly detection,
industrial damage detection, image processing, anomaly detection in text data, and sensor networks [12]. A data mining approach has proven its effectiveness in extracting knowledge from sequences of data stored in a time-series databases over repeated measurements of time [41]. Data acquired from WSNs is usually periodic and is stored in a centralized database once it reaches the base station, hence can qualify as inputs to such data mining systems. Later in this chapter, we propose a centralized abnormal node detection mechanism in which a sensor node is classified as either normal or abnormal depending on its sensor readings collected at the base station within a certain time window (offline).

Recently, researchers have explored the effectiveness of a number of data mining techniques to extract knowledge from real-time data streams which dynamically flow in and out of an observation platform, and are usually characterized as always changing, requiring a quick response time, and allowing only one scan over a given stream [2, 41]. This is very beneficial in occasions where the stream data points cannot fit in memory as is the case with the resource-constrained sensor motes. Once again, sensor data can be seen as streams transmitted by one sensor node and later received by and flow in and out of the neighboring nodes. Mining sensor data streams is the process of extracting knowledge from sensor streams using single-scan, on-line, and multilevel stream processing and analysis methods [41]. In the next chapter, we propose an on-line data-centric anomaly detection mechanism whereby the mining/detection task is carried out by every node in the network before possibly aggregating data from child nodes. It uses a stored hypothesis to determine the class of the node—either itself or its child node in case the sensor network is deployed in a hostile environment—as either legitimate or abnormal. The hypothesis is obtained during the training phase whereby nodes extract statistical features on the fly and
communicate them to the base station rather than communicating the raw sensor readings. One major advantage of this approach is that stream data points are discarded rather than being stored in memory.

In this section, we discuss some of the popular existing data-centric anomaly detection mechanisms which were proposed over the past decade. Figure 5.1 shows a taxonomy of data mining (DM) based schemes for anomaly detection in sensor networks. All these schemes have their roots in machine learning (ML). Some rely on classification for distinguishing normal from anomalous behaviors while others use clustering techniques. Classification is a supervised learning problem in which a classifier (i.e. model, learner, or hypothesis) is constructed describing a pre-labeled sensor dataset (training set). The model can then be used to classify future unlabeled sensor data tuples. Since sensor data is usually unlabeled in advance, domain expertise can be leveraged to label sensor dataset generated in earlier deployments, or synthetic anomalies can be injected into existing datasets along with their class labels using the anomaly models proposed in Chapter 4. Clustering, on the other hand, is an unsupervised learning technique which does not require the sensor training dataset to be labeled in advance. With clustering, sensor data tuples will be grouped together into a number of groups and each group is assigned a class label (the simplest case generates two clusters; one labeled as normal while the other is labeled as anomalous). The classification of schemes in Figure 5.1 is based on the type of the machine learning technique being used. The following discusses each category and pinpoints the strengths and/or weaknesses of the schemes in each of these categories. We also discuss alternative approaches to anomaly detection in sensor networks at the end of the section.
Rajasegarar et al [105] proposed a \( k \)-nearest-neighbor (\( k \)-NN) clustering-based off-line approach for anomaly detection in sensor networks. More concisely, there are two forms of clustering involved: (i) a fixed-width distance-based clustering algorithm [29] with width (diameter) \( w \) and, (ii) a \( k \)-NN clustering algorithm which runs on the base station to detect anomalies. The fixed-width clustering algorithm may simply run directly on the base station (centralized) after collecting all raw sensory data from all sensor nodes in the network, or the process can be distributed among the sensor nodes to eventually form global clusters at the base station. Figure 5.2(a) illustrates this process. To perform the clustering in a distributed manner, each sensor node applies the fixed-width clustering algorithm locally over the data features to form a number of distinct clusters. This number is determined algorithmically and data features are a transformed and normalized form of raw sensor data samples collected over a certain time window. To obtain such features, each sensor node first computes a tuple of local statistics (mean and variance for transformation, min and max for normalization) applied over the time window for each attached sensor and then
forwards the tuple to the base station. The base station then computes the global statistics tuple by combining all received statistical parameters and then floods the resulted tuple to the entire network eventually arriving at each sensor node. After applying fixed-width clustering, each sensor node forms a cluster summary for each cluster consisting of the centroid and the number of samples in the cluster. It then forwards all cluster summaries to the parent node (assuming a tree topology as in the figure). Upon receiving all cluster summaries from all of its children, a parent node merges the received clusters with its own clusters based on the inter-cluster width (if inter-cluster distance between the centroids of two clusters is less than the width $w$) and then forwards the new cluster summaries up the tree. This process continues until the base station receives all cluster summaries from its children, merges the clusters with its own clusters, and applies $k$-nearest-neighbors distance-based clustering to identify anomalous clusters. In $k$-NN, each sensor node chooses $k$ shortest distances among the distances to other clusters (inter-cluster distances). The average of the $k$ shortest inter-cluster distances ($ICD_i$) of each sensor node $i$ is then calculated and compared with the total number of inter-cluster distances ($ICD$). A cluster is considered normal if its $ICD_i$ value lies within one standard deviation from the mean $ICD$, otherwise, it is marked as anomalous.

Simulations over the Great Duck Island dataset have shown that this approach can achieve 4% false positive ratio while drastically reducing the amount of energy consumed when the decentralized approach is in play (due to collecting cluster summaries instead of raw data). This clustering-based approach, however, has several deficiencies. First, anomaly detection is always performed off-line at the base station. Second, there is a communication overhead involved due to the need for collecting local statistics from each
sensor node to compute the global statistics tuple at the base station and then disseminates it over the network to reach all sensor nodes. Third, decentralized clustering is performed independently from the sensor data collection, therefore the base station can still be able to detect anomalies in the network but lacks any raw or even aggregated sensor data. Finally, the number of clusters $k$ in the $k$-NN clustering algorithm and the cluster width parameter $w$ are user-specified design parameters and their values can dramatically affect the detection accuracy. For instance, if $w$ is chosen to be at one of the extremes, the algorithm cannot detect anomalies at all.

Another clustering-based mechanism was proposed by Rajasegarar et al in [103] which leverages the unique shapes of a hyper-ellipsoid (see Figure 5.3). In this work, each sensor node stores a number of observations over a certain time window. Each observation includes a number of sensing modalities. Each node then computes the sample mean and the covariance matrix of these observations. It then takes the inverse of the covariance matrix to form a hyper-ellipsoid. The resulting hyper-ellipsoidal parameters (i.e., sample
mean and inverse covariance matrix) are then used to find the Mahalanobis distance between all observations and the sample mean. If the latter is greater than a certain threshold, the sensor observation at a time is considered a local anomaly, otherwise, it is locally normal. After this, a node communicates its elliptical parameters as well as the number of its samples to its parent node. A parent node merges its children hyper-ellipsoidal parameters with its own parameters, and communicates the results up the tree. The gateway node then communicates the globally computed hyper-ellipsoid over each detection window back to all the nodes in the network where a node would find global anomalies by computing the Mahalanobis distance, once again, and comparing it with the threshold.

The above clustering-based scheme incurs relatively high computational complexity bounded by the cost of computing the inverse of the covariance matrix. The scheme also introduces an additional overhead due to communicating the hyper-ellipsoidal parameters up and down the tree. In addition to the computation and communication cost incurred by
this scheme, it requires all sensor readings within a large time window to be stored in memory. Referring to the same example in [103], storing 3000 data points of 5 sensing modalities each requires less than 30100 bytes (assuming the size of a sensing modality value is 2 bytes) large enough not to fit in the data memory of the most popular resource-constrained sensor platforms (e.g., MicaZ uses 4 KBytes of RAM).

**SVM-based**

An SVM-based anomaly detection mechanism in sensor networks was proposed in [102]. Similar to [66], the mechanism uses quarter spheres as the separating hyperplane and a one-class (unsupervised) SVM. Each sensor reading \( \{x_i : i = 1..n\} \) is mapped to its corresponding feature vector in a higher dimensional space using a non-linear mapping function \( \phi(x_i) \). After the mapping is complete, the anomaly detection considers solving the following SVM optimization problem:

\[
\min_{R \in \mathbb{R}, \xi \in \mathbb{R}^n} \quad R^2 + \frac{1}{vn} \sum_{i=1}^{n} \xi_i \\
\text{subject to} \quad \|\phi(x_i)\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0, \quad i = 1..n
\] (5.1)

Both the slack variables \( \{\xi_i : i = 1...n\} \) and the regularization parameter \( v \in (0, 1) \) control the number of anomalous feature vectors. \( R \) is the radius of the quarter sphere. In other words, the detection algorithm finds the best quarter sphere that has the minimum radius from the origin while covering most of the mapped feature vectors. To reduce computational complexity, the primal problem (5.1) can be mapped to the following dual linear problem using Lagrange and the kernel trick:
\[ \min_{\alpha \in \mathbb{R}^n} \quad -\sum_{i=1}^{n} \alpha_i k(x_i, x_i) \]

subject to \[ \sum_{i=1}^{n} \alpha_i = 1, \]
\[ 0 \leq \alpha_i \leq \frac{1}{vn}, \quad i = 1..n \]

where \( \alpha_i \geq 0 \) are Lagrangian multipliers and \( k(x_i, x_i) = \phi(x_i)\phi(x_i) \) is the kernel function which is equivalent to \( R^2 \). In order to be able to solve equation (5.2), the feature vectors need to first be centered by subtracting from each feature vector the mean of all feature vectors in the space. This results in the centered feature vectors and the new norms \( \tilde{k}(x_i, x_i) \) will be extracted form the diagonal elements of the kernel matrix \( \tilde{K} = K - \frac{1}{n}K - K1_n + \frac{1}{n}K1_n \), where \( K = k(x_i, x_j) \) is the original \( n \times n \) kernel matrix and \( 1_n \) is an \( n \times n \) matrix with all values equal to \( 1/n \). Once the Lagrange multipliers \( \alpha_i \) are computed, an anomalous feature vector is detected if its corresponding \( \alpha_i \) equals to \( \frac{1}{vn} \).

As in [105], the anomaly detection algorithm can be entirely applied at the base station (off-line) after collecting all raw sensor data, or it can be executed in a distributed manner (on-line). Figure 5.2(b) illustrates the operation of the distributed anomaly detection algorithm. Each leaf node (assuming a tree topology) applies the above anomaly detection approach locally (i.e. maps sensor data to feature vectors, solves equation (5.2) for the Lagrangian multipliers \( \alpha_i \), and marks any \( \alpha_i \) with the value \( \frac{1}{vn} \) as a local anomaly) and finds the local ‘minimum’ radius \( R \) and the L-2 norms of the feature vectors \( \tilde{k}(x_i, x_i) \) (i.e. temporal correlation or distance between each measurement and the rest of \( n \) measurements). It then stores these two values in its memory and sends its local radius to its parent node. After receiving all local radii from its children, a parent node computes the
mean, median, maximum, or minimum radius which will serve as the global radius $R_m$. A parent node then communicates the global radius back to its children. Upon receiving the global radius, a feature vector is identified as *globally* anomalous by the child node if its norm $\tilde{k}(x_i, x_j) > R_m^2$. Depending on the desired coverage of global anomaly detection, global radius computation can be done at any sensor node in the network. As an example, in Figure 5.2(b), both nodes $C$ and $S$ can compute a global radius and communicate it with their children and their children’s children. The figure shows a tree topology, however, the algorithm can adapt to any other topology assuming there exists a leader node that can compute the global radius and communicates it with its neighbors.

Similar to [105], the decentralized anomaly detection algorithm was tested on the Great Duck Island dataset and was able to reduce energy consumption several time folds due to the reduced communication cost of exchanging radius information between nodes instead of propagating the raw sensor readings. Moreover, the decentralized algorithm scales well since anomaly detection is performed locally by each sensor node over the node’s measurement window. The main disadvantage of this approach is the high complexity imposed by solving the dual linear problem of equation (5.2). It is also expensive to perform matrix computation resulting from centering the feature vectors. It is therefore very costly for a low-power resource-constrained sensor node to carry out such computations. Moreover, there exists a significant detection delay due to the time needed to receive the computed global radius at each node. Finally, there are several user-defined parameters that need to be carefully chosen; these include the type of the distance-based kernel function and its parameters, the regularization parameter $v$, and the computation criteria of the global radius (e.g. mean, median, minimum, or max).
Zhang et al [144] extends the above SVM-based technique to be able to distinguish between sensor faults and interesting events while running in real-time (i.e. detection is performed at every sensing interval) using spatio-temporal correlations among neighboring nodes. The mechanism assumes that nodes are fully synchronized. Rather than unicasting the local radius $R$ to its parent, instead, each node locally broadcasts its learned radius to its neighbors. Upon receiving local radii from all neighboring nodes, each node then finds the median radius $R_m$. A single measurement is considered an outlier if its distance from the origin of the feature space is larger than both $R$ and $R_m$. Once an anomaly is detected, the anomalous node requests its neighbors distances from their own origins. If its own distance is larger than its local radius and the median distance of all neighbors is larger than the median radius (and both are larger than the global radius), the anomaly may be classified as an event. Otherwise, it may be considered a measurement fault. In addition to this, the authors also proposed three techniques to update or re-learn the normal model over time. Compared to the work of Rajasegarar et al [102], all proposed techniques achieved better detection rate with less false positives.

**Bayesian-based**

A number of Bayesian based approaches to the problem of anomaly detection in sensor networks have been proposed [28, 46, 51, 64, 112]. In these works, a Bayesian model is learned online or off-line and then used for inferencing anomalies. For instance, in [28], a naive Bayesian model is first learned on the fly using only the nodes readings and then its accuracy is tested over time. Once the accuracy improves significantly, the learned predictor is put in action; the reading is considered anomalous if the number of misclassi-
Modification instances is higher than half. The latter scheme highly relies on the fact that the entire observation history of the sensor node is summarized by the previous reading and the node’s immediate neighbors (parent and child). This assumption holds if the sensor readings are both spatially and temporally correlated. The scheme uses $m$ data ranges to predict the missing reading or replace its detected anomalous reading with a more likely reading within the predicted data range. In order to estimate the probabilities of the Bayesian model, a total of $(1 + m + 3m^2/2 + m^3/2)$ counters are needed to be stored at each node. If the phenomena is stationary over space, the learning is performed in-network whereby each node communicates its local counters to its parent after every specified epoch and then resets its counters to zero. The parent node, in turn, sums up its children counters with its own and forwards the resulting counters up the tree. This process continues until the base station receives the last counter summaries and accumulates them over a number of epochs.

In [112, 113], the authors proposed a 5-state supervised Hidden Markov Model (HMM) based approach where the states correspond to day, night, short faults, noise faults, and constant faults. The latter HMM is a simple dynamic bayesian network (DBN) which is characterized by (i) the set of observations (i.e., sensor readings), (ii) the five states, (iii) the emission probabilities of obtaining a sensor observation given the current state, (iv) the state transition probabilities, and (v) the initial state probabilities. Due to high variations in the sensor readings (e.g., as a result of calibration errors), temperature observations were discretized using bins of size $0.1^\circ C$ each. For each sensor observation, the classifier generates the most likely state that resulted in the current sequence of observations. If a fault state is encountered, the observation is classified as anomalous. This scheme can both detect and classify faults according to their types.
The major disadvantage of Bayesian-based approaches is the complexity involved in estimating the model parameters (i.e. probabilities) in terms of both processing and communication costs. For example, it takes $O(N^2T)$ to find the transition probabilities for the HMM based scheme using a forward-backward procedure \cite{100} where $N$ is the number of states and $T$ is the number of observations. In addition, the stationary version of the naive Bayes approach in \cite{28} does not scale well as the packet size communicated to the base station and containing the model parameters (or counters) increases with the increase of the number of model classes (number of data range intervals).

**Neural Net based**

Artificial neural networks (neural nets) are desired due to their lower computational cost compared to other machine learning mechanisms. There has been few proposed anomaly detection mechanisms for sensor networks which rely on neural nets \cite{15,117,129}. A neural network is a weighted directed graph connecting input vertexes (called neurons) to output vertexes via a number of vertexes called hidden neurons. To learn the model, back-propagation is usually used whereby the output of each neuron is computed using the edge weights and the neurons’ outputs from the previous layer. The result is then multiplied by a nonlinear activation function.

An Echo State Network (ESN) based online fault detection scheme was proposed in \cite{15} in which all neurons are interconnected and can have a zero-weighted edges. The connections and weights of the neurons are generated randomly and do not change, reducing the learning algorithm to a simple linear regression. Only the output weights change during training and the activation function used is the hyperbolic tangent. After learning
the model at the base station (sensor readings are communicated to the sink during training), the ESN classifier is deployed onto the sensor mote. Anomalies are identified on the fly if the distance from ESN predictions to the actual sensor readings is larger than a certain threshold. The main disadvantage of this scheme is that memory usage, detection time, and detection accuracy do not scale well with the number of neurons. In terms of memory, a larger ESN model requires larger program memory space to store the weight matrix \( O(n^2) \) where \( n \) is the number of neurons) and the activation function. The authors attempted to reduce the size of the weight matrix by using compressed raw storage which only stores the 10% non-zero weights and the matrix layout \( O(2n_z + n + 1) \) where \( n_z \) is the number of neurons with non-zero output weight). They also reduce ROM usage by using a tan-like activation function which offers a high approximation of the original tanh function. The same holds for detection speed; a larger number of neurons requires longer time to identify anomalies (from about 0.5 seconds with 50 neurons to 4 seconds with 400 neurons). The scheme was able to detect short and noise faults within the LUYF (Life Under Your Feet) dataset [84] with a lower false positive compared to simple rule-based schemes [112].

Siripanadorn et al [117] proposed an unsupervised neural nets based scheme for detecting faults in sensor network deployments. The scheme employs a Self-Organizing Map (SOM) as the classifier coupled with Discrete Wavelet Transforms (DWT) to reduce the amount of communicated sensor readings to half without losing the significant feature of the data. SOM maps high-dimensional data onto low-dimensional (typically two-dimensional) discretized representation of the input space which is, in this case, the wavelet coefficients representing the fault-free data. Similar to Chang et al [15], a new observation is considered anomalous if the distance between the reading and the predicted output neu-
ron is greater than a certain threshold. In addition to synthetic datasets, the latter scheme was tested on three real-world datasets (IBRL, SensorScope, and NAMOS datasets) and could detect 100%, 83%, and 99% of faults in the three datasets, respectively. As in [15], this scheme is sensitive to the number of neurons. In addition, the number of training epochs needs to be carefully chosen.

**Ensemble-based**

Recently, Curiac and Volosencu [17] have exploited ensemble classifiers in detecting data-centric anomalies in sensor networks. The proposed mechanism which leverages spatio-temporal correlations uses 5 different dynamic hypothesis to tackle detection: (i) a simple average classifier, (ii) an autoregressive linear predictor, (iii) a neural net, (iv) a neural net autoregressive predictor, and (v) an adaptive Neuro-Fuzzy Inference System (NFIS) classifier. Each of these classifiers exploits a certain feature (spatial redundancy, temporal redundancy, spatio-temporal correlation, and recent temporal changes). The final label prediction of the node, to determine if it is either anomalous or normal, is based on a weighted vote of the five hypothesis. The mechanism was tested using a mixture of simulated data and data from a real world deployment. Erroneous temperature readings were simulated by applying heat from a heat lamp or cold air from a mobile air conditioning unit. The ensemble was able to detect 99.71% of the temperature anomalies. In this mechanism, there is a good number of threshold values that need to be determined a priori. Moreover, certain classifiers may impose higher complexity such as neural net and NFIS classifiers. Finally, classifiers’ weights need to be carefully specified based on simulations and experiments.

Yu et al [141] proposed an intrusion detection system based on SLIPPER; a rule-based
boosting algorithm. The authors defined some rules for detecting data-centric anomalies. However, the effectiveness of the proposed IDS was never tested by any means.

**Other Approaches**

Besides leveraging data mining techniques, other research has explored other mechanisms to detect data-centric anomalies in sensor networks [9, 36, 63, 106, 113, 139]. Next, we briefly discuss three of these schemes. For a list of other techniques, the reader is referred to [143].

**Rule-based.** In this approach, a set of heuristic rules or constraints is predefined (learned) based on scientific experience, domain knowledge, and/or analysis of datasets from existing and earlier deployments [52, 59, 106, 113]. A typical rule will have a threshold parameter to identify faults or anomalies when the data does not satisfy this parameter. An example of rule-based anomaly detection was presented in [106]. In their work, the authors defined a set of rules (SHORT, NOISE, BROKEN, BAD BATTERY SENSOR, and INVALID NLDR) each associated with a threshold parameter for filtration of faults from normal data. The filtration process was applied over a dataset obtained from a recent deployment which took place in Bangladesh and aimed at measuring the quality of groundwater in Ganges Delta. During this process, each sensor data tuple was assigned a bit vector that corresponds to all pre-defined rules (when one rule applies, its corresponding bit is set). Based on the applied rules, the system assigned an ordered list of human actions and if several rules applied, the system reported the rule that applied to most tuples from the same sensor. Similarly, in [113], two major rules are defined: (i) the NOISE rule and; (ii) the SHORT rule. By using the NOISE rule, a node is assumed to experience a NOISE
fault if the standard deviation of the sensor readings within a time window is above a certain threshold and a SHORT fault if the rate of change between two consecutive readings is above a certain threshold. CONSTANT faults are detected when the standard deviation is zero. The rules were applied over multiple datasets obtained from four real-world deployments (IBRL, GDI, SensorScope, and a deployment in Lake Fulmor at James Reserve (NAMOS)) identifying SHORT and NOISE faults with different rates of false positives and false negatives. Rule-based techniques are simple which makes them suitable for being implemented at the sensor level. However, such mechanisms highly depend on the threshold parameters. In addition, with some specific rules, other parameters such as the window size is hard to specify and can only be determined by domain experts.

**SSA-based.** Yao et al [139] proposed a Segmented Sequence Analysis (SSA) approach to anomaly detection which leverages spatio-temporal correlations to construct a piecewise linear model of the sensor data stream. The scheme aligns the latter model with a reference model and then computes the distance between both models using a similarity measure. An anomaly is detected if the distance measure is greater than a specified threshold which needs to be carefully determined. This scheme requires humongous amount of memory (max 2052 KBytes) and takes 5 seconds on average to process about 30,00 samples.

**Statistical-based.** In [9], a simple and practical statistical-based data-centric anomaly detection algorithm was presented, whereby each sensor node maintains and periodically updates a probability distribution of difference between its current reading and both its previous reading as well as its neighbors’ readings. A reading is flagged as anomalous using a simple significance test in which the current difference is compared to a certain...
threshold (i.e., significance level). The statistical distribution parameters are updated on the fly, so previous differences are dropped at every new epoch. This reduces the size of information stored at the node. Two methods were proposed to update the distribution of differences: (i) a parametric approach where differences are well fit by a known distribution (e.g., Gaussian distribution with mean $\mu$ and variance $\sigma^2$), and (ii) a non-parametric approach using simple frequency histograms. The memory cost of the former method is only $2(k + 1)$ floating-point numbers where $k$ is the number of neighbors. The non-parametric method increases the memory usage to $m(k + 1)$ where $m$ is the number of bins in the histogram. The proposed algorithm was tested over a synthetic dataset as well as a sensor dataset obtained from an earlier sensor network deployment at the Sevilleta LTER site [24]. It was able to detect measurement faults in the temperature readings with entirely no false positives. Similar to rule-based and SSA-based techniques, this algorithm requires human expertise to determine the threshold value. In addition, the time to learn the distribution model still requires about 9 hours in typical scenarios.

Table 5.1 summarizes the anomaly detection mechanisms discussed in this section. In the next section, we propose a novel data-centric anomaly detection framework which can be categorized under ensemble-based schemes. The proposed framework has shown to be effective in identifying compromised data behaviors and measurement faults in real-world sensor datasets. It improves the detection accuracy using a popular boosting algorithm called AdaBoost [35].
<table>
<thead>
<tr>
<th>Scheme</th>
<th>Technique</th>
<th>Type</th>
<th>Operation</th>
<th>Centralized</th>
<th>Decentralized</th>
<th>GDI</th>
<th>HRL</th>
<th>SensorScope</th>
<th>SAMOS</th>
<th>LUYF</th>
<th>Bangladesh</th>
<th>Synthetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rajasegarar '06 [105]</td>
<td>k-NN Clustering</td>
<td>Unsupervised</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rajasegarar '10 [103]</td>
<td>Ellipsoidal Clustering</td>
<td>Unsupervised</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rajasegarar '07 [102]</td>
<td>Quarter sphere SVM</td>
<td>Unsupervised</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bealearey '09 [104]</td>
<td>Basis Function</td>
<td>Supervised</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shama '08 [112]</td>
<td>HMM</td>
<td>Supervised</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chang '09 [117]</td>
<td>Echo State Networks</td>
<td>Supervised</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Siripanadorn '10 [117]</td>
<td>SOM-DWT</td>
<td>Supervised</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Curiac '12 [46]</td>
<td>Ensemble</td>
<td>Supervised</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yu '08 [141]</td>
<td>SLIPPER</td>
<td>Supervised</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ramanathan '10 [106]</td>
<td>Rule-Based</td>
<td>Other</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sharma '09 [112]</td>
<td>Self-Inspected</td>
<td>Other</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Le '07 [148]</td>
<td>SOM-based</td>
<td>Other</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brittensier '07 [9]</td>
<td>Statistical-based</td>
<td>Other</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.2 Adaboost-Based Centralized Data-Centric Anomaly Detection in Sensor Networks

In this section, we propose a novel data-centric anomaly detection framework rooted in machine learning which accurately identifies both measurement faults in real-world sensor datasets as well as compromise data behaviors. The schemes discussed in the previous section were not tested against compromised data behaviors which are more likely to occur in sensor networks deployed in hostile environments. Similar to measurement faults, compromise data behaviors hinder the fidelity of the collected sensor data. Let us consider a network of a number of MicaZ motes (see Table 1.1) deployed in an uncontrolled hostile area. MicaZ motes are accompanied by connectors to a programming board which enables developers to install the application code before deployment or at site with the help of a battery-powered laptop. By using inexpensive off-the-shelf equipments such as AVR JTAG ICE [81], an adversary may attach a laptop to one of the motes [42] and downloads the entire ROM code containing all cryptographic keys within seconds. Once compromised, a sensor node is reprogrammed to emit forged data to mislead the base station. Therefore, conventional cryptographic techniques [57,93] become ineffective and additionally impose
a heavy load on such resource-constrained devices.

The proposed framework, which we call ABANDON, leverages the AdaBoost algorithm [35] as its learning engine. As a supervised learning approach, AdaBoost has the following advantages: (1) It has a good theoretical basis; (2) The implementation is simple; (3) The computational complexity is not too high; (4) It rarely suffers from the over-fitting problem; and (5) It easily utilizes heterogeneous categorical and continuous data types. AdaBoost has been proven effective in detecting network intrusions in wired networks [47] using the KDD Cup 1999 dataset. In the KDD Cup 1999 dataset [20], each network connection is characterized by several features. Unlike the latter work, we extract simple statistical features that highly express the anomalous behavior of a sensor node. Such features have proven to be very effective in identifying a data-centric anomaly.

5.2.1 System Overview and Assumptions

We consider a sensor network model consisting of \( M \) sensor nodes deployed in an area of interest. Sensor nodes report their readings—potentially over multiple hops—to a base station. The network may also be structured according to a tiered-architecture [90] whereby few number of PC-like sensor devices such as Stargate [121] act as masters while the rest of the nodes are highly resource-constrained. Each master node receives sensor readings from a number of neighboring nodes and is assumed to be honest. In a real-world deployment, sensor nodes have multiple sensors that are capable of sensing more than one phenomenon of interest. We refer to a phenomenon as a sensing modality. We assume that, to achieve the highest performance, all sensor nodes are spatially and temporally correlated. In other words, sensing modalities provide close values in space and time. If the latter assump-
tion holds, all features extracted over the sensing modalities can be used for classification whereas a sensing modality providing uncorrelated data might not be preferable unless it is combined with other extracted features. For existing ecological or environmental deployments, this assumption is valid due to two reasons [9]: (i) node-to-node spacing is usually less than 100-200 meters, and (ii) sensing modalities such as air temperature, relative humidity, light flux, soil moisture, and soil temperature exhibit a substantial amount of temporal coherence due to common climate drivers. Furthermore, in a case of a two-tier architecture, the network is assumed to be densely populated (i.e., higher neighborhood density). This is essential for obtaining larger training sets at the master nodes to avoid over-fitting. ABANDON runs on the base station or the master nodes. Every node is characterized by a series of statistical features which are inherently different from the features used by boosting algorithms when detecting intrusions in a wired network. Figure 5.4 illustrates the two major steps taken before a sensor node is identified as either anomalous or normal. Like most data mining based frameworks, our proposed framework employs two major steps: (i) feature extraction, and (ii) classification. During feature extraction, a number of statistical features are extracted which characterize the nature of the node’s sensor readings over a certain time window $T_p$. The feature vector is then fed to the classification stage where a previously trained AdaBoost ensemble classifier predicts the node’s class as either anomalous or normal.

### 5.2.2 Feature Extraction

The success of unsupervised methods in this area [102, 103, 105, 144] inspired us to utilize statistical features to characterize the behavior of the sensor data. As discussed in previous
Figure 5.4: Detecting anomalies with ABANDON

chapters, measurement faults exhibit unique data behaviors that deviate from the normal readings in a certain way. Similarly, compromised nodes send falsified readings in an effort to mislead the querying node. A naive compromised node may always send falsified data, whereas some smart compromised nodes can send mixed data containing bogus and correct values or some values that are slightly deviated from the true value.

Nevertheless, raw data sent by sensor nodes cannot reliably characterize anomalous data behaviors. Therefore, unsupervised methods use statistical measures or summaries (such as variance, etc.) over the raw sensor readings to evaluate the quality and trustworthiness of such readings. Hence, we seek a number of features that are highly expressive of the various types of data-centric anomalies. Ni et al [88] suggested a number of statistical features that can be utilized for detecting measurement faults. These include rate of change, spatial distance, mean, variance, temporal correlation, and spatial correlation. Some of these features have already proven their effectiveness in statistical-based [135] and rule-based detection techniques [113]. Some of these features may also highly characterize the nature of a compromised data behavior. Inspired by these facts, we leverage a number of statistical features in detecting both sensor data faults (e.g., outliers, spikes, stuck-at
faults, noise faults) as well as compromise data behaviors (naive, smart). Some of these features are perfect for detecting certain types of data-centric anomalies than the other.

We now explain how features are being extracted. Suppose $S = \{1, 2, ..., M\}$ is the set of sensor nodes administered by the base station or one of the master nodes. Within a certain time window $T_p$, the master node (or base station) collects $L$ readings of dimension $K$ from each administered node $m \in \{1, ..., M\}$. Here, $K$ represents the number of sensing modalities ($M >> K$). For each sensing modality $k \in \{1, ..., K\}$, the master node extracts a feature vector $\{f_{1,k}, f_{2,k}, ..., f_{N,k}\}$ over the node’s readings $\{x_{l,k} : l = 1...L\}$ where $N$ is the number of desired features. The final feature vector for all sensing modalities is represented as $f = \{f_{1,1}, f_{1,2}, ..., f_{1,K}, f_{2,1}, f_{2,2}, ..., f_{2,K}, ..., f_{N,1}, f_{N,2}, ..., f_{N,K}\}$. Four types of features may be extracted. The first feature highly characterizes most of the data-centric anomalies and is calculated as

$$
    f_{n,k} = \sqrt{\frac{(x_{1,k} - \sigma)^2 + (x_{2,k} - \sigma)^2 + ... + (x_{L,k} - \sigma)^2}{L}}
$$

where $\sigma = \frac{x_{1,k} + x_{2,k} + ... + x_{L,k}}{L}$. This feature is actually the standard deviation of sensor readings over $T_p$ for sensing modality $k$. The standard deviation feature is effective in identifying naive and smart compromise behaviors as well as certain measurement faults (e.g., noise faults).

Once again, let us consider for a specific sensing modality $k$ during a fixed time interval $T_p$, $L$ readings are collected by the master node from each administered node $m$. Assuming that sensor readings are collected every epoch (say 30 seconds), and only those readings within the time window $T_p$ are considered by the master node during extraction, we can
compute the absolute deviation of node $m$ among all nodes’ $i$th readings as

$$ad_{i,k} = |x_{i,k} - \mu|$$  \hspace{1cm} (5.4)$$

and $\mu = \frac{\sum_{M}(x_{i,k})}{M}$. Notice above that sensor nodes are not necessarily synchronized and some readings may be missing as a result of possible network contentions. If one reading is missing within the extraction window $T_p$, it is simply excluded from calculations. Given all absolute distances for all readings over $T_p$, we can now compute the second type of features:

$$f_{n,k} = \frac{\sum_{i=1}^{L} ad_{i,k}}{L}$$  \hspace{1cm} (5.5)$$

This feature, indeed, captures the spatial correlation between each node’s readings and the readings of its neighbors. The absolute distance feature may capture compromised data behaviors that utilize an offset parameter to achieve highly deviated readings from the true phenomenon but with rather very small and stealthy gradient over time. The spatial distance to the neighboring nodes can also be utilized to identify events within a small region.

Two other features may be utilized in identifying a certain type of data-centric anomalies. Consider again $L$ sensor readings of sensing modality $k$ collected from node $m$, the following equation captures the data range over the extraction window $T_p$:

$$f_{n,k} = \max[x_{1,k}, x_{2,k}, \ldots, x_{L,k}] - \min[x_{1,k}, x_{2,k}, \ldots, x_{L,k}]$$  \hspace{1cm} (5.6)$$

The data range feature, which is simply the difference between the highest and lowest readings observed during $T_p$, can be effective in detecting different types of measurement
Table 5.2: Summary of Extracted Statistical Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Symbol</th>
<th>Correlation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Deviation</td>
<td>SD</td>
<td>Temporal</td>
<td>The variance of readings within $T_p$</td>
</tr>
<tr>
<td>Data Range</td>
<td>DR</td>
<td>Temporal</td>
<td>The difference between highest and lowest reading within $T_p$</td>
</tr>
<tr>
<td>Highest Gradient</td>
<td>MG</td>
<td>Temporal</td>
<td>The maximum rate of change between two consecutive readings over $T_p$</td>
</tr>
<tr>
<td>Absolute Distance</td>
<td>AD</td>
<td>Spatial</td>
<td>The absolute deviation of a sensor reading from its neighbor’s readings</td>
</tr>
</tbody>
</table>

faults, events, and malicious data behaviors. Finally, if the master node or the base station
keeps track of the maximum rate of change of a certain sensing modality over one extraction
window, it may detect outliers as well as spike faults in that specific sensing modality.
Notice that the standard deviation, data range, and maximum gradient features capture the
temporal correlation among sensor readings overtime. Table 5.2 lists all four features that
can be utilized by our framework.

5.2.3 Ensemble Classification

Rule-based and statistical-based detection techniques highly depend on the threshold value.
In some cases, it is difficult to determine the optimal threshold. In addition, in a typical
environmental monitoring application, a number of sensing modalities is being measured and
hence imposing different threshold requirements. To avoid adjusting threshold values,
clustering-based detection mechanisms employ a distance measure such as Euclidean or
Mahalanobis distance to identify anomalies. Despite the fact that unsupervised learning
methods do not require any pre-labeling mechanisms, they have their own weaknesses as we discussed in Section 5.1. Nevertheless, supervised learning based mechanisms usually
achieve higher detection accuracy compared to their unsupervised counterparts provided a
common ground truth. In Chapter 4, we were able to design very highly expressive
anomaly models which serve as the ground truth in this case. Thus, our framework adopts
a supervised learning algorithm called AdaBoost into detecting data-centric anomalies.

AdaBoost [35] is a machine learning algorithm that can be used in conjunction with many other learning algorithms to improve their performance. Consider a two-class problem with the output variable \( y \in \{-1, 1\} \). Given a feature vector \( f \), a classifier \( G(f) \) returns a class prediction of \(-1\) or \(1\). For \( U \) input feature vectors \( F = f_1, f_2, \ldots, f_U \) and the corresponding true classification sequence \( Y = y_1, y_2, \ldots, y_U \), the error rate over the entire training sample for this classifier \( G(f) \) is

\[
\bar{\epsilon} = \frac{1}{U} \sum_{i=1}^{U} I(y_i \neq G(f_i))
\] (5.7)

and the expected error rate on future predictions is \( E_{FY} I(Y \neq G(F)) \). We choose \( U \) feature vectors rather than \( M \) to increase the size of the training dataset, and in some cases to avoid over-fitting (\( U = \lambda M \), where \( \lambda \) is the number of extraction windows). A classifier is called a weak classifier if its error rate \( \bar{\epsilon} \) is slightly better than random guessing. AdaBoost sequentially applies the weak classification algorithm to repeatedly modified versions of the data producing a series of weak classifiers \( G_q(f), q = 1, 2, \ldots, Q \). The final prediction is then computed as the weighted majority vote of all weak classifiers:

\[
G(f) = \text{sign} \left( \sum_{q=1}^{Q} \alpha_q G_q(f) \right)
\] (5.8)

where \( \alpha_1, \alpha_2, \ldots, \alpha_q \) are computed by AdaBoost according to the following equation

\[
\alpha_q = \log \left( \frac{1 - \epsilon_q}{\epsilon_q} \right)
\] (5.9)

In this work, we use decision stumps as well as decision trees as weak classifiers. De-
Algorithm 1 AdaBoost.M1

**Input:** Sequence of $U$ labeled feature vectors $(f_i, y_i), y_i \in \{1, -1\}, i = 1, ..., U$
Base classifier $G(f)$ (e.g., decision stump, decision tree)
Number of iterations $Q$

1: Initialize: $w_i = 1/U, i = 1, 2, ..., U$
2: for $q = 1$ to $Q$ do
3: Call $G_q(f)$ using $w_i$
4: Get back a hypothesis $h_q: \{f_{1,k}, f_{2,k}, ..., f_{N,K}\} \rightarrow y_i \in [-1, 1], i = 1, 2, ..., U$
5: Calculate error $\epsilon_q$ of $h_q$ as:
6: $\epsilon_q = \frac{\sum_{i=1}^{U} w_i I(y_i \neq G_q(f_i))}{\sum_{i=1}^{U} w_i}$
7: Compute $\alpha_q = \frac{1}{2} \ln((1 - \epsilon_q)/\epsilon_q)$
8: Set $w_i \leftarrow w_i \cdot \exp[\alpha_q \cdot I(y_i \neq G_q(f_i))], i = 1, 2, ..., U$
9: end for
10: Output: $G(f) = \left[\sum_{q=1}^{Q} \alpha_q G_q(f)\right]$

dcision trees [99] are tree-like graphs where intermediate nodes act as decision points and branches represent the decision outcome at the branching node. The final prediction appears at the leaf nodes (i.e., class labels). Decision stumps are one-level decision trees with one internal node (root) connected directly to the leave nodes. A decision tree or decision stump algorithm looks at all possible thresholds for a specific feature vector $f$ extracted over the sensor data of each node, and selects the one with the maximum information (classification) gain (i.e., the one that best partitions the set of feature vectors $f_1, f_2, ..., f_U$ into two sets; one with class label $y = 1$ and the other is labeled with $y = -1$). AdaBoost proceeds as in Algorithm 1 [35, 43, 110].

5.2.4 Evaluation

Depending on the type of data-centric anomaly, a feature extractor may choose to extract different set of features. In case of uncertainty, choosing the entire set of four features will not influence the classification error since AdaBoost is robust to ‘useless’ features. In this section, we test the effectiveness of the ABANDON framework in identifying measurement
Detecting Anomalies in IBRL dataset

In this experiment, we extract 3 statistical features (SD, DR, and MG) which capture a number of measurement faults prevalent in the IBRL deployment such as spike faults, outliers, stuck-at faults, and noise faults. We will show that such features are adequate to identify all types of faults in this dataset with entirely no false positives in some cases. Since light readings in the IBRL dataset did not exhibit a strong spatial correlation (some motes were potentially located in darker areas than others), we avoided using the spatial distance feature in detecting faults.

Sharma et al. [113] had tested their proposed algorithms using the IBRL dataset as well. They first tested the effectiveness of enforcing simple rules such as the NOISE rule which states that “if the standard deviation of the sensor readings over a certain time window is above a certain threshold, the sensor is assumed to suffer a noise fault”. In [113], noise faults in the IBRL dataset are those we refer to in Chapter 3 as the combination of spike, stuck-at, and noise faults that usually appeared at the end of the deployment (i.e., those caused by low battery). Further, in [113], a basic 2-state Hidden Markov Model (HMM) based mechanism was used to detect both noise faults and outliers in the temperature readings. Given a sequence of readings, a HMM provides the state that most likely resulted in the current observation. That being said, if the state associated with a new reading is a NOISE state, the reading is classified as faulty. In order to use the HMM approach, it is required to first estimate its parameters. This is achieved during the training phase whereby
faults are injected into the training dataset. This approach proved to have higher detection accuracy with lower false positives and false negatives than the other two techniques discussed in [113] (i.e. rule-based, time-series based).

We evaluated the effectiveness of our approach in detecting sensor data faults prevalent in the IBRL dataset by first using a portion of the existing faults as ground truth. Then we injected a number of fault models designed in Chapter 4 to obtain a ground truth highly representative of the actual faults in the IBRL dataset.

**Training With Real Faults.** In this experiment, we trained the classifier with real faults which already existed in the IBRL dataset. We considered a number of time windows where few nodes exhibit one or more faults. For example, referring back to Figure 2.4, the one-day time window starting at 4:27 AM in March 8 and the 8-hour window starting at 12:29 PM in March 18 can both be good candidates for training as they cover a good portion of a spike fault exhibited by nodes 18, 19, and 35, and possibly followed by a stuck at fault. Recall that both time windows start at the time when the node’s temperature/humidity sensor obtains a faulty reading due to the low voltage level of the node. For noise faults, any time window (> 120 epochs) between March 23 and March 26 will be sufficient to represent a noise fault exhibited by node 6 (refer back to Figure 3.6(b)).

After that, we measured the detection accuracy by testing the classifier using a different subset from the IBRL deployment that potentially contains combinations of spike, stuck-at, and noise faults. We used 40 time windows of different lengths (30mins – 8hours) and ran the extraction and classification phases a number of times over the same time window. At each run, we used different number of statistical features extracted over the time frame of temperature, humidity and light readings, as well as different number of faulty nodes to
lead to either the training or the testing sets. For 70% of the time windows, a combination of only two or three decision stumps was adequate to identify all the spike faults (followed by a stuck-at fault) and noise faults that have not been explored yet. 30% of the time windows resulted in higher false positives and false negatives. This is due to the length of the time window being very small or very large. If the window size is very small, it is not adequate to cover a spike behavior, while a larger time window covers a portion of the readings from other nodes that are included in the initial training or testing set of anomalous nodes.

A conclusion which suggests that training a classifier using real-world existing sensor datasets is tricky and error prone, and relying on injected fault models highly representative of the sensor data is less sensitive to human errors. Moreover, it is difficult to determine the optimal length of the time frame where sensor faults may be present. To measure the
effectiveness of the classifier to identify all types of faults while at the same time requiring no human intervention in determining which features should be extracted, we extracted all three statistical features (temporal deviation, highest gradient, and data range) concurrently over all sensing modalities and fed them to the classification stage. Figure 5.5 plots the detection accuracy over a few number of iterations during one of the experimental runs. Combining all features may result in higher detection error. However, in some cases the number of false positives and false negatives eventually degrades to zero after a few number of iterations. Using a combination of features extracted over a number of sensing modalities can capture higher number of faulty behaviors in a real-world sensor dataset and at the same time can still eliminate false positives (and false negatives) entirely.

As for outliers, we trained the classifier with 3 outlying values from Table 4.1. Not surprisingly, all other outliers were easily detected with zero false positives using a combination of two simple decision stumps.

**Training With Injected Faults.** In this experiment, we injected a number of faults at different time windows during the beginning of the IBRL deployment. This guarantees that most nodes are healthy and have not exhibited faults due to a low battery. Before injecting faults, however, the list of few known outliers in Table 4.1 was first filtered out especially if the outliers lie within the injection window. This ensures that only the injected faults are present in the training dataset, and hence improving our confidence in the obtained ground truth. The injection model was designed as follows. The first 400 epochs (200 minutes) were allocated for outliers. We randomly injected 10 outliers into the temperature and humidity readings of 3 nodes. After that, we used the same nodes to inject spike faults followed by stuck-at faults into a window of 3500 epochs of the node’s tem-
perature and humidity readings. Finally, we proceeded with only one node and injected a noise fault into its temperature readings over a duration of 800 epochs.

Training was conducted in a similar manner as with real faults, specifying different lengths of the time window for each run. During the testing phase, we chose time frames where real outliers are present in the IBRL dataset. As expected, all outliers in Table 4.1 where easily detected using a combination of only two simple decision stumps. After that, we tested the classifier over time windows containing spike behaviors (potentially followed by stuck at behaviors as well as noise faults). Once again, such faults were detected using a combination of few decision stumps and depending on the window length, false positives varied from 0.0208 to entirely no false positives. For smaller window sizes (< 4 hours), the ratio of false negative was higher than that of the false positive (< 0.0417). Both ratios, however, maintained steady after a number of AdaBoost iterations. Finally, we achieved similar error ratios as in Figure 5.5 when using all statistical features combined to detect faults within the temperature and humidity readings. The errors might fluctuate differently over the first few number of iterations. However, both false positives and false negatives eventually decreased to zero. This shows that the anomaly models well represented the true behavior of real-world anomalies and hence resulted in a solid reference ground truth. That is said, automatic injection of anomalies according to the models designed in the previous section eliminated any human errors and tedious work associated with manual visual inspections.
Figure 5.6: The first two hours of training data during the experimental deployment. The black lines correspond to readings from normal nodes.

Table 5.3: Compromise Data Behaviors (Training Phase)

<table>
<thead>
<tr>
<th>Node ID</th>
<th>Sensing Modalities</th>
<th>Deviation</th>
<th>Anomaly Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>light, acoustic</td>
<td>$\sigma = {20, 30}$</td>
<td>naive</td>
</tr>
<tr>
<td>12</td>
<td>temp, light, acoustic</td>
<td>$\sigma = {30, 30, 30}$</td>
<td>smart</td>
</tr>
<tr>
<td>17</td>
<td>temp, light, acoustic</td>
<td>$\sigma = {10, 10, 10}$</td>
<td>naive</td>
</tr>
<tr>
<td>20</td>
<td>temperature</td>
<td>$\sigma = 15$</td>
<td>smart</td>
</tr>
</tbody>
</table>

**Detecting Compromise Data Behaviors**

In addition to measurement faults, we also tested the effectiveness of the proposed framework in detecting compromised nodes. We designed an experimental testbed consisting of 25 MicaZ motes with different kinds of uncalibrated sensor boards (MTS300 and MTS310). The network was deployed in a $5 \times 5$ grid and employed a multi-hop routing protocol whereby most sensor motes communicate their sensor readings every 30 seconds to the base station over multiple hops. Hence, learning is performed at the base station. The sensor motes collected room temperature, light intensity, and acoustic signals. Since no calibration was performed, there was a significant amount of drift in the sensor readings. The experiment consisted of two phases, the training phase and the testing phase. During the training phase, 84% of the nodes were programmed to behave normally, whereas the code of the rest was modified to simulate behaviors of compromised nodes. Table 5.3 lists the compromise parameters during the training phase while Figure 5.6 plots such behaviors.
over the first two hours of the experiment. Notice the high variations in the light readings among the nodes. Despite our efforts to achieve high spatial correlation within the light fluxes, the sensors were exposed to slightly different levels of light coming from the ceiling lambs in the lab. To this end, temporal variations were more effective in classifying nodes and adding the absolute distance to neighboring nodes did not increase the classification error after all. Nevertheless, the ensemble classifier was able to leverage spatial correlations among the temperature and acoustic readings as well as temporal coherences of the three sensing modalities.

During the testing phase, we maintained the same percentage of compromised nodes. However, we selected other nodes to be compromised and varied their compromised behaviors randomly based on the affected sensing modalities, the degree of deviation, and the compromise category (naive vs smart, see Table 5.4). Figure 5.7 plots the sensor readings of the misbehaving nodes as well as sensor readings from the rest of the network. Notice

Table 5.4: Compromise Data Behaviors (Testing Phase)

<table>
<thead>
<tr>
<th>Node ID</th>
<th>Sensing Modalities</th>
<th>Deviation</th>
<th>Anomaly Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>light, acoustic</td>
<td>$\sigma = {10, 15}$</td>
<td>naive</td>
</tr>
<tr>
<td>5</td>
<td>light, acoustic</td>
<td>$\sigma = {30, 30}$</td>
<td>smart</td>
</tr>
<tr>
<td>10</td>
<td>temp, light, acoustic</td>
<td>$\sigma = {10, 8, 10}$</td>
<td>naive</td>
</tr>
<tr>
<td>17</td>
<td>temp</td>
<td>$\sigma = 15$</td>
<td>smart</td>
</tr>
</tbody>
</table>
the true outlier in the acoustic data. We did not filter out this single outlier and hence it ac-
counted for the higher rate of false positives when acoustic features were extracted over the
period within which it was present. Moreover, in order to diversify the compromise behav-
iors, two nodes where intentionally selected to inject falsified readings of a unidirectional
behavior using a varying noise (node 12 during training and node 5 during testing).

We ran the experiment several days and extracted several testing sets. We then fed
the sets into the classification stage and calculated the error rates. Here, we considered
the first two hours of both the training and testing experiments (Figures 5.6 and 5.7). We
varied both the length of the extraction window \( T_p \) and the number of extraction windows
\( \lambda \) and extracted two features (SD and AD) over the three sensing modalities. We also
considered a simple decision stump and a decision tree as the base classifier (i.e., weak
learner). Figure 5.8(a) illustrates the accuracy of the ensemble classifier after 100 iterations
using decision stumps as base classifiers. The learner always achieves the highest accuracy
with different values of \( \lambda \) when \( T_p \) is set to 10 minutes. The degradation in accuracy at
higher values of \( \lambda \) owes to the acoustic outlier present in the testing set since increasing \( \lambda \n
may reach a point where one extraction window would cover the outlier. The false positive
rate when \( \lambda = 5 \) is 0.008 which is the proportion of single outlier to a set of extracted 125
samples (feature vectors). Figure 5.8(b) shows the total rate of false positives and false
negatives for this particular run when the extraction window was set to 10 minutes. Finally,
the classification accuracy decreases for \( T_p \) larger than 10 minutes. In addition, over larger
\( T_p \), the number of extraction windows \( \lambda \) should be set large enough to avoid over-fitting.
Figure 5.9 shows an instance where over-fitting may take place (when \( \lambda \) is set too small
while the length of the extraction window is large (25, and 30 minutes). To avoid over-
(a) The accuracy of the ensemble classifier using decision stumps. The accuracy is maximized when $T_p$ is set to 10 minutes.

(b) Respective errors when the extraction window is set to 10 minutes. False positives are due to the single outlier in the acoustic signal.

Figure 5.8: Accuracy of the ensemble classifier after 100 iterations using decision stump as the base classifier.

fitting, one could also set the number of iterations to a small number limiting the ensemble from over-training itself [98]. Cross validation can also be used to determine the best number of iteration automatically [35]. In our experiment, four iterations were adequate to maintain a test error of 4% for all $T_p$ periods. This also reduces the computational complexity of the classifier.

To further improve the accuracy of the classifier for larger $T_p$, we chose decision trees as the base classifiers. Figure 5.10(a) plots the classification accuracy in this case. Using decision trees can reduce the classification error or totally eliminate it at the cost of increased complexity. For example, comparing the classification accuracy in the latter figure when $T_p$ is set to 15 minutes to those shown in Figure 5.8(a), one can notice that the classification error improves by a factor of 5 when $\lambda$ is set to 4 while setting $\lambda$ to 2, 3, or 6 eliminates the classification error which was introduced when using decision stumps as the base classifier. Using trees as weak classifiers also avoided the over-fitting cases that where introduced when using decision stumps with higher values of $T_p$ (see Figure 5.10(b)).
Figure 5.9: Classification accuracy at 100 iterations. The learning process suffers from over-fitting for larger $T_p$ when $\lambda$ is set too small.

Figure 5.10: Accuracy of the ensemble classifier after 100 iterations using decision trees as base classifiers.

As a final evaluation, we compare AdaBoost (using decision stumps as the base classifiers) with two variants of a Support Vector Machine (SVM) classifier: (i) the max ‘soft’ margin linear discriminant classifier which performs better in cases where extracted features are linearly inseparable (SVM SOFT); and (ii) the dual form of the former (SVM DUAL).
SVM classifiers are another type of supervised machine learning algorithms that have a strong theoretical basis as we discussed in Section 5.1. Table 5.5 lists the test errors of the evaluated classifiers. The first three classifiers incorporate both the SD and AD features whereas AdaBoost₁ only uses the temporal standard deviation (SD) feature.

Table 5.5: Test Errors of The Evaluated Classifiers

<table>
<thead>
<tr>
<th>$T_p$ (minutes)</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM SOFT</td>
<td>0.08</td>
<td>0.08</td>
<td>0.05</td>
<td>0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>SVM DUAL</td>
<td>0.06</td>
<td>0.08</td>
<td>0.11</td>
<td>0.05</td>
<td>0.06</td>
</tr>
<tr>
<td>AdaBoost₂</td>
<td>0.04</td>
<td>0.01</td>
<td>0.05</td>
<td>0.08</td>
<td>0.07</td>
</tr>
<tr>
<td>AdaBoost₁</td>
<td>0.04</td>
<td>0.04</td>
<td>0.08</td>
<td>0.04</td>
<td>Overfit</td>
</tr>
</tbody>
</table>

From the table, we can see that AdaBoost results in a higher detection accuracy compared to SVM in cases where the extraction window is small, whereas, both versions of SVM perform better than AdaBoost for larger $T_p$ most of the times. This is because these SVM versions are designed for linearly inseparable data and as $T_p$ increases there is more chance that an anomalous behavior taking place earlier matches that of the current phenomena since sensing modalities (e.g. temperature) may vary overtime. Also notice that using an additional feature may improve the accuracy of the AdaBoost classifier but not necessarily. This depends on $T_p$, the phenomena being sensed, and the anomalous behaviors being monitored. In our experiment, it helps to extract spatial distance features as compromised nodes emit readings that are deviated from the readings of normal nodes (either intermittently or at all times).
5.3 Summary

In this chapter, we first surveyed a number of data-centric anomaly detection schemes which have their roots in machine learning and data mining. We also explored few alternative approaches to the problem of identifying data-centric anomalies in sensor networks. After presenting the pros and cons of each of these schemes, we proposed a novel data-centric anomaly detection framework which leverages boosting techniques as its learning engine. The proposed framework has proven effective in detecting different types of data-centric anomalies. It accurately identified measurement faults within the IRBL dataset with no false positives in most cases. We also ran a real-world experimental deployment at our lab and used the proposed approach to detect compromise data behaviors. It was shown that classification accuracy improves when using a short extraction periods (10 minutes). Furthermore, using decision trees as base classifiers may improve the accuracy of the ensemble for larger windows.
Chapter 6

Practical Online Data-Centric aNomaly Detection (POND)

The ABANDON algorithm proposed in the previous chapter runs at the base station or at the Stargate-like master nodes in a multi-tier sensornet deployment. A tremendous amount of effort is required to allow such algorithm to run at the sensor level. Designing such a system brings with it a new set of challenges. First, sensor motes are highly resource-constrained. A PC-class memory storage is far from being possibly installed at these motes. In addition, powerful micro-controllers that drive robots and other embedded devices still incur a high cost and are not preferable as drivers for sensor motes; the original vision of a sensor network deployment constitutes a large number of low-cost sensor devices to measure their environment with the least cost possible. And on top of all, 'energy' is scarce. Communicating a large number of unnecessary packets, performing CPU-exhaustive operations, and writing on an external flash is overly avoided in most sensornet protocols. Second, low-cost sensor motes usually run light weight operating systems that have their own set of limitations and are programmed using a compressed version of a typical programming language where not all functionalities are supported. To this end, we propose a practical online data-centric anomaly detection (POND) framework that employs light-
weight, accurate, and rapid detection scheme at every sensor node in a sensor network deployment.

This chapter is organized as follows. In the following section, we present the main motivations behind designing a generalized online data-centric anomaly detection framework for sensor network deployments. Section 6.2 describes an overview of our POND framework and lists realistic assumptions. Section 6.3 discusses the details of each functional block in the POND framework. Section 6.4 elaborates on few implementation aspects. We evaluate the framework in section 6.5 through simulations and a real-world indoor experimental deployment. Simulations enabled us to carry out evaluation on a bigger scale in an effort to measure the effectiveness of our framework when employed in large-scale sensor network deployments, whereas a smaller scale real-world deployment showed the applicability and practicality of the proposed framework to function in real-world. In the same section, we present our developed SNMiner toolkit which offers a great amount of functionality for running our experiments. Section 6.6 explains limitations and possible evasions to our framework. Finally, we give a general discussion and present some future work in Section 6.7, and conclude in Section 6.9.

6.1 Motivations

Let us consider a hybrid event-driven query application where nodes only report their sensor readings upon detecting a specific event (e.g., once detecting a particular situation about fire-alert from humidity readings, sensor nodes periodically report humidity and wind data in order to closely measure the possibility of a fire [91]). In the latter application, a detection mechanism should be employed to first detect the desired event and later on, during
sensor data collection, identify and possibly filter out measurement faults which may lead to falsified conclusions. As another example, consider a query-based sensornet application deployed in a hostile environment. In this type of applications, it is important to achieve precise and sensitive interpretations about the monitored region. In an effort to mislead interpretations over the collected data, an adversary may inject falsified readings into the network. At the same time, sensor nodes may suffer from connection failures or low battery level that could lead to anomalous behaviors different from those compromised ones. Hence, in this type of applications, the detection mechanism should be capable of identifying and possibly filtering out both types of anomalous behaviors towards protecting the quality and trustworthiness of the collected sensor data. VigilNet [44] serves as a perfect example of a hybrid application. In VigilNet, 70 sensor nodes work together to detect an intruding vehicle (event). To accurately estimate the speed and location of an intruder, faulty readings should be filtered out by fusing (i.e., aggregating) readings from a number of adjacent nodes. Due to the nature of the VigilNet application, sensor nodes are also vulnerable to being compromised. Hence, an anomaly detection scheme should be able to identify malicious data behaviors, measurement faults, as well as events of interest. Some of the existing approaches are efficient in detecting measurement faults but may not necessarily apply to events or malicious data behaviors of potentially different characteristics. If a node is running its own detector as in [103] and [28], a compromised node will not be detected as adversary can entirely manipulate the detector’s code. In addition, most of the recently proposed schemes [103,105,113] rely on datasets obtained from an earlier sensornet deployment and have not been employed in a real-world deployment (i.e., their applicability in real world sensor network deployments is not clear). Usually
deployments form a tree topology similar to the topology view of Figure 6.1 and apply a number of collection (e.g., CTP [37], MultihopLQI [97]) and configuration dissemination protocols. Therefore, an online anomaly detection mechanism should run alongside these protocols without disrupting the normal operation of the sensornet.

In this chapter, we propose an online practical anomaly detection framework to identify data-centric anomalies in sensornet deployments. Our framework may run alongside sensor data collection and is tunable to accommodate different sensornet applications. The framework enables application administrators to train a network of deployed sensors, instructs the nodes to extract on-line statistical features, and leads every node in the network to carry out the classification.

Atop challenges faced by wireless networks, a sensor network mainly imposes two additional resource constraints: memory and battery constraints. Our framework uses on-line feature extraction whereby statistical features are being extracted on the fly overcoming the need to maintain a list of the sampled data points over the extraction window. In addition to memory constraints, energy is the most precious resource perhaps in every sensornet.
deployment and the major abuser of the energy resource is communication. Therefore, our anomaly detection framework also attempts to minimize the amount of communication by only communicating the statistical summaries at the initial deployment (i.e., during the training phase) and later on, if the sensornet is being deployed in a hostile environment, may communicate these summaries between the child nodes and their parents. If a sensor node is designed to hold an external flash of an adequate size, writing on flash memory is still very expensive in terms of power consumption and is avoided during feature extraction. In addition, the latter process is conducted in a way whereby nodes sample sensor readings faster than the reporting frequency. Hence, POND is capable of rapidly and accurately identifying data-centric anomalies in existing sensor network deployments “on the fly” while reducing memory and communication overhead.

6.2 System Overview and Assumptions

We consider a sensor network deployment consisting of $M$ sensor nodes and each node can observe $A$ sensing modalities. The proposed framework aims at environmental monitoring applications where motes sense basic phenomena (e.g. temperature, humidity, light intensity, etc.). While more complex phenomena can be considered, the advantage of the high detection speed offered by the POND framework is lost. This is due to the high energy involved when sampling complex sensors such as seismic and image sensors [101]. If by any means sampling the sensors faster than the reporting frequency incurs a high non-negligible energy cost, feature extraction can be performed over the sensor readings obtained at the original reporting frequency. The latter modification leads to reporting only one legitimate packet over a significantly larger extraction window (i.e., the last packet). This, however,
should not be an issue since sensor readings are assumed to be spatially and temporally 
correlated, and reporting one packet that summarizes the sensed phenomena over the ex-
traction window can be adequate. The network employs a multi-hop tree-like collection 
protocol such as CTP. Furthermore, our mechanism aims at but not limited to large deploy-
ments [82] since nodes in densely populated areas are more likely to be correlated in space 
and time. Given a large number of sensor nodes, the global correlation assumption can also 
be relaxed by using a clustering technique to merely group the correlated sensors in dif-
ferent clusters and apply the anomaly detection within each cluster. Figure 6.2 shows two 
main phases that constitute the POND framework; the training/re-training phase and the ac-
tual on-line detection phase. During the training phase, anomalous data behaviors of a set 
of selected misbehaving nodes are first injected into the sensor network. A node’s behavior 
can be characterized by different parameters depending on the desired anomalous behav-
iors to be learned. Every node periodically extracts a statistical feature vector describing its 
sensed data and communicates it up the tree leveraging the same tree structure used for data 
collection. Upon receiving the set of all feature vectors from all nodes in the network, the 
base station runs a number of classification algorithms to obtain the best detector and then 
disseminates the resulting detector code down the tree to reach every sensor node in the 
network. During the online detection phase, each node continues to incrementally extract 
a feature vector. Based on the application in hand, a sensor node will discard anomalous 
readings if the sensornet is deployed for monitoring applications, sends an alarm signal to 
the base station in case of an event in event-driven applications, or communicates its feature 
vector up the tree if the sensornet was deployed in a hostile environment.
6.3 The POND Framework

In this section, we detail the description of each functional block in the training and online detection phases of the POND framework.

6.3.1 Misbehavior Injection

At the base station, every node is labeled as either anomalous or normal node and then a number of anomalous behaviors is injected into the network. This can be done in two ways: off-line at the gateway node or in-network. While monitoring the sensor network, the end user may inject the desired anomalous behavior into the sensor stream of an anomalous node upon receiving its readings at the gateway node. This simple approach does not add any extra overhead to the deployed network as all the tedious work is done on the PC side (i.e., off-line). However, since sensor readings are usually reported at a low frequency, this slows down the training time. A better alternative is to instruct all nodes in the network to sample their readings faster than the reporting frequency as we will discuss later (e.g., 10 times faster). However, by doing this, communicating these samples will add a noticeable
overhead. Off-line injection is adequate if feature extraction is performed entirely at the gateway node. For a more realistic scenario and to enable on-line feature extraction during the training phase, we may need to deliver the anomaly models in-network. There are several ways to achieve in-network delivery or anomaly models. One could think of using a point-to-point routing protocol such as AODV or Dymo to deliver single anomaly models to individual nodes. From a practicality point of view and as we use TinyOS-powered devices for evaluation, we wish to leverage existing implementations of network protocols. There are several implementations of AODV in TinyOS, and Tymo is the TinyOS implementation of Dymo that we can use. Like every reactive routing protocol, Dymo floods the network with route requests to discover the desired route to the anomalous node and requires each node to maintain a routing structure. A second way is to rely on the link estimates and parent information in CTP to construct a global view of the tree topology off-line. Then, one can generate a reverse path to the intended node by creating a source-routing like packet. Source routing was also implemented in the newer versions of TinyOS and can be utilized for this purpose. Using this approach, the packet size increases with the increase of the depth of the tree. A third way is to leverage the tree structure of Koala and sending anomaly models hop-by-hop down the tree to reach the individual nodes. Koala, however, requires logging of sensor readings to flash memory and is not the de facto standard for TinyOS-powered sensornet applications. One last approach is to disseminate the anomaly models to the entire network and those nodes that are labeled as anomalous set their local data model correspondingly. A number of configuration dissemination protocols are already being used in existing deployments (e.g. DRIP, DIP [72]). Upon receiving the list, a node marks itself as anomalous only if its id is in the list and accepts the behavioral
Table 6.1: A Summary of Anomaly Models

<table>
<thead>
<tr>
<th>Anomaly Type</th>
<th>Mapping function $\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlier</td>
<td>$\phi = c_k$ where $c_k \in C_k$</td>
</tr>
<tr>
<td>Spike</td>
<td>$\phi = \theta(\beta_{0,k} + \beta_{1,k}q_k + \beta_{2,k}q_k^2 + \ldots + \beta_{r,k}q_k^r)$, $q_k = (p_k - n)$, $\theta()$ is a rounding function</td>
</tr>
<tr>
<td>Clipping</td>
<td>$\phi = c_k$ where $c_k \in {c_1, c_2}_k$</td>
</tr>
<tr>
<td>Compromise</td>
<td>$\phi = x_{p,k} \pm \sigma_{p,k}$ where $\sigma_{p,k}$ is the variance of a white Gaussian noise with zero mean</td>
</tr>
<tr>
<td>Event</td>
<td>$\phi = o_k + x_{p,k} \pm \sigma_{p,k}$ where $o_k$ is the data offset</td>
</tr>
</tbody>
</table>

parameters assigned to it. Each of the four approaches discussed imposes an extra overhead. The choice among the four approaches depends on the anomaly type and space. In general, source routing, although not scalable, is suitable for injecting single fault models into a small number of nodes, whereas it imposes less overhead to disseminate the anomalous behaviors if the number of anomalous nodes is large (e.g., 35% compromised nodes).

As discussed in Chapter 4, each type of anomalies may require different model parameters and mapping function expressed by Equation 4.1. Table 6.1 summarizes the anomaly models discussed in Chapter 4, some of which will be used in the evaluation section. In the evaluation section, we inject a combination of measurement faults, compromise data behaviors, and events to measure the effectiveness of our framework in detecting various co-existing anomalies.

6.3.2 On-line Feature Extraction

After the anomaly injection phase, every node periodically extracts a statistical feature vector describing its sensed data every extraction window $T_p$. Each sensor node periodically obtains a new sample of its sensing modalities every $T_s$ ($T_p = \beta \times T_s$, where $\beta$ usually ranges between 5 and 10 to avoid high sampling overhead, we will show the effect of
such value on the overall energy cost in section 6.5) and incrementally computes a statistical summary $f = \{f_{1,1}, f_{1,2}, \ldots, f_{1,K}, f_{2,1}, f_{2,2}, \ldots, f_{2,K}, \ldots, f_{N,1}, f_{N,2}, \ldots, f_{N,K}\}$, where $N$ is the number of desired statistical features and $K \subset A$ is the number of desired sensing modalities. Ni et al [88] have suggested several important statistical data features that can be effective in detecting a fault such as the rate of change, spatial distance, mean, variance, temporal correlation, and spatial correlation. These can also be effective in case of an event or a malicious data behavior. In this work, temporal features such as highest gradient or incremental variance are extracted by performing time-series incremental analysis over the sensor stream whereas spatial features such as distance require information about the node’s neighborhood prior to extraction. For each sensor node to determine the set of desired features, a bit vector may be disseminated over the network during the injection phase or hard-coded in the node’s code image. The following discusses the set of online features which we proved to be effective in detecting anomalies as well as other features that may be utilized in some cases.

**Temporal Deviation**

To incrementally compute the standard deviation feature $n$ of the sensing modality $k$, each sensor node keeps track of three values [2]: (i) the number of readings $L$ over the extraction window $T_p$; (ii) the sum of readings $\sum x_{l,k}$ where $\{x_{l,k} : l = 1 \ldots L\}$ are the sensor readings of modality $k$; and (iii) the square sum of readings $\sum (x_{l,k})^2$. The following equation is then used to calculate the standard deviation feature $n$ for each modality $k$:

$$f_{n,k} = \sqrt{\frac{\sum (x_{l,k})^2 - (\sum x_{l,k})^2}{L - 1}}$$ (6.1)
The temporal deviation feature is effective in identifying naive and smart compromise behaviors as well as certain measurement faults (e.g., noise faults).

**Highest Gradient**

This feature is utilized to identify outliers and spike faults. It keeps track of the maximum rate of change over one extraction window. In this case, we only need to maintain two values; the highest (max) rate of change and the sensor reading from the previous epoch or sample \( x_{t-1,k} \). The incremental feature extraction of the highest gradient feature \( n \) for each modality \( k \) proceeds as in Algorithm 2. The highest gradient feature may also be effective in identifying events.
Algorithm 2 Highest Gradient Feature Extraction

1: \( f_{n,k} = x_{2,k} - x_{1,k} \)
2: for all \( \{x_{l,k} : l = 3...L\} \) in \( T_p \) do
3:     if \( |x_{l,k} - x_{l-1,k}| > f_{n,k} \) then
4:         \( f_{n,k} = |x_{l,k} - x_{l-1,k}| \)
5:     end if
6: end for

Algorithm 3 Data Range Feature Extraction

1: \( \text{max} = \text{min} = x_{1,k} \)
2: for all \( \{x_{l,k} : l = 2..L\} \) in \( T_p \) do
3:     if \( x_{l,k} > \text{max} \) then
4:         \( \text{max} = x_{l,k} \)
5:     else
6:         if \( x_{l,k} < \text{min} \) then
7:             \( \text{min} = x_{l,k} \)
8:         end if
9:     end if
10: end for
11: \( f_{n,k} = \text{max} - \text{min} \)

Data Range

Data range is an effective feature which captures the difference between the highest and lowest readings within \( T_p \). It can be incrementally calculated using the pseudocode of Algorithm 3. The data range feature can be effective in detecting different types of measurement faults, events, and malicious data behaviors.

Neighborhood Distance

The spatial distance to the neighboring nodes can be utilized to identify events within a small region as well as compromise data behaviors. It captures the spatial correlation between each node’s readings and the readings of its neighbors. To compute such feature off-line, the base station is required to know the global view of the network. Fortunately, a CTP data packet contains the parent and link quality information within its header. There-
fore, we can utilize such information to build the tree topology view of the network. We may then estimate the closest $c$ neighbors from every sensor node and use their readings at a time instance (or small time window) to find the distance between the sensor node’s reading and the average reading of its neighbors. To obtain the sensor readings at the time of training, a node may append the average of its readings over the extraction window within the payload of the feature vector. At the sensor level, during the actual online detection phase, neighborhood readings can be obtained by snooping on the data traffic within the node’s vicinity. Usually, the extra energy spent during snooping is negligible. The average of the neighborhood readings can be incrementally computed without the need to save the actual readings from the neighbors. At the end of the extraction window, a node then computes its neighborhood distance feature by simply taking the difference from its own incremental average of $T_p$ to the incremental neighborhood average. This feature is effective in detecting events.

Other Features

Temporal deviation, highest gradient, and data range features can all be good indications of a CONSTANT fault. All three features combined can be used to detect a CONSTANT fault since the value of all three features will be zero over the extraction window $T_p$. An alternative to the latter feature combination is a feature that is set to a high value if any of the latter three statistics computed over one extraction window sums to zero. Moreover, there are several other features that can be utilized. However, they may not be as effective as the latter features in detecting anomalies (i.e., may not improve on the classifier’s accuracy). Examples include temporal autocorrelation, simple moving average (SMA) over
$T_p$, moving average of the voltage level, and the CUMulative SUM (CUSUM) for change detection.

### 6.3.3 Feature Collection

If we denote the data reporting frequency as $T_c$, then $T_c = \lambda \times T_p$. $\lambda$ represents the number of contiguous extraction windows. Usually the reporting frequency is about 5-30 minutes in environmental monitoring applications. In order to leverage the same tree structure used for data collection during the training phase, every node sends its extracted feature vector up the tree every $\lambda$. To avoid collision with other feature vectors from the sibling nodes, a node waits a random back-off (few milliseconds) before sending its feature vector at the end of every extraction window. Figure 6.3(a) shows a scenario where $\lambda = 2$ and $\beta = 10$. Recall that $\beta$ is the number of readings being sampled during the extraction window $T_p$.

The number of feature vectors necessary to construct a classifier at the base station can vary. To reduce communication overhead and to rapidly re-train a new detector (e.g., an application administrator decides to incorporate new events), setting $\lambda$ to 2 or 4 may still generate a predictor with very low false positives as we saw in Chapter 5. It can also be set higher than that merely for performing cross validation on a larger set of feature vectors and/or avoiding over-fitting. In the evaluation section, setting $\lambda$ to one was adequate to achieve zero false positives in some cases with no instances of over-fitting.

### 6.3.4 Filtering Noise

Recall that labels are known by the application administrator whenever retraining is to be initiated. During retraining, existing anomalous behaviors (faults, events, or known malicious data behaviors) will be filtered out at the base station by the off-line detector learned
previously (it is a copy of the same detector running on each node after dissemination as we will discuss later in this section). Recall that during the training phase, sensor nodes communicate feature vectors describing their data. If sensor node $i$ is sampling faulty readings, the offline detector will filter all readings from node $i$ regardless of its pre-assumed label. Filtering noisy data enables the base station to generate a more accurate classifier. During training at the initial deployment, however, the noise filter is disabled since no detector will yet be learned. Hence, we assume that during the initial deployment all sensor nodes are benign or normal. Otherwise, we may need to use a classification algorithm that considers noise within the training dataset such as RobustBoost.

### 6.3.5 Classification

Upon receiving the statistical summaries from all sensor nodes, the base station evaluates a number of supervised machine learning algorithms and chooses the algorithm that gives the best trade-off between complexity and accuracy (i.e., small false positives and false negatives). For rapidly detecting compromise data behaviors within a small $T_p$, we have proven in Chapter 5 that boosting algorithms (we used AdaBoost) are superior to other classification algorithms such as Support Vector Machines (SVMs) and simple decision trees. AdaBoost can also be effective in classifying other data-centric anomalies as well (e.g., measurement faults). It essentially generates a final hypothesis that is a linear combination of weak classifiers whose error rates are slightly better than random guessing. Several classification algorithms can serve as the base classifier. A simple decision stump, which chooses the feature that has the maximum information gain first, is desirable due to its low complexity. To further reduce the complexity associated with boosting and to generate an
easier-to-interpret ensemble classifiers, we use an alternating decision tree (ADTree) [34]. Algorithm 4 shows the steps necessary to generate an ADTree classifier that is a generalization of voted decision stumps. \( W_+ (d) \) and \( W_- (d) \) represent the total weights of the positive and negative examples, respectively, which satisfy condition \( d \). \( \alpha \) is a smoothing factor used to avoid zero frequency problems [94]. Each rule \( r \) in the tree consists of a condition \( d \), a precondition, and two predictions \( a \) and \( b \). The first rule \( r_0 \) always holds (i.e., \( d \) is always \textit{true}) and thus has only one prediction value \( a_{\text{root}} \). Notice that, in the case of decision stumps, all rules in the final rule set \( R \) have preconditions that will evaluate to \textit{true}. Also, note that a rule in a consecutive iteration is very likely to have a similar condition as that of a rule obtained in previous iterations (lines 11-15). In this case, only the predictions of the existing rule will be updated using the new rule’s prediction values. This avoids the burden of choosing the optimal value of \( Q \) (number of iterations). Choosing this value has been an open research problem and sometimes the best values is determined through cross validation [34]. In the evaluation section, we loosely choose 300 iterations as the value of \( Q \) and still obtain very low complex ADTree classifiers without overfitting.

6.3.6 Classifier Dissemination

Over The Air (OTA) programming is often used in sensornet deployments to reprogram the sensor nodes without physically attaching each node to a programming board. This is essential in large scale deployments of unattended sensors for two reasons: (i) convenience (no physical human presence is needed); and (ii) due to the difficulty of accessing the deployed nodes in some regions. OTA programming protocols such as Deluge [49] can be used to disseminate the new code image containing the predictor (e.g., ADTree), which is
Algorithm 4 ADTree with decision stump as base classifier

Input: Labeled vectors \((f_i, y_i), y_i \in \{1, -1\}, i = 1, ..., U\)
Number of iterations \(Q\)

1: Initialize: \(\epsilon = 0.05, w_{i,0} = 1, i = 1, 2, ..., U\)
\(\alpha = \epsilon + \sum w_{i,0}\)
\(a_{\text{root}} = \frac{1}{2} \ln \left( \frac{W_+(\text{true}) + \alpha}{W_-(\text{true}) + \alpha} \right)\)
Add rule \(r_0\) with condition \(true\) and prediction \(a_{\text{root}}\) to a final set of rules \(R\)

2: Pre-adjust: \(w_{i,1} = w_{i,0} e^{-ay_i}\)
\(\alpha = \epsilon + \sum w_{i,1}\)

3: for \(q = 1\) to \(Q\) do
4: Generate the set \(D = (f_n, k < \theta)\) using \(w_{i,q}\)
5: for all \(d_i \in D\) do
6: \(Z_q = 2 \left( \sqrt{W_+(d_i)W_-(d_i)} - \sqrt{W_+(\neg d_i)W_-(\neg d_i)} \right)\)
7: end for
8: Select \(d_q\) that minimizes \(Z_q\)
9: Calculate predictions of \(r_q\):
10: \(a = \frac{1}{2} \ln \left( \frac{W_+(d_q) + \alpha}{W_-(d_q) + \alpha} \right), b = \frac{1}{2} \ln \left( \frac{W_+(\neg d_q) + \alpha}{W_-(\neg d_q) + \alpha} \right)\)
11: if \((r_q\) with a similar condition \(d_q\) is in \(R)\) then
12: Add its predictions to existing rule’s predictions
13: else
14: Add \(r_q\) to \(R\) along with its new predictions
15: end if
16: \(w_{i,q+1} \leftarrow w_{i,q} e^{-r_q(f_i)y_i}\)
17: \(\alpha = \epsilon + \sum w_{i,q+1}\)
18: end for

19: Output: \(\text{class}(f) = \text{sign} \left( \sum r_i(f) \right)\)

the output of the classification phase. Since Deluge is inherently reliable, each node will receive a copy of the modified image and reboots itself. The modified code may reset the behaviors of those nodes that were selected as anomalous nodes during the misbehavior injection phase to normal.

6.3.7 On-line Detection

When a node receives the predictor, it transforms itself to the detection phase and stays in that phase until instructed by the base station otherwise. A node continues to extract a feature vector every \(T_p\) and feeds it to the predictor. The predictor may produce both false
positives and false negatives if new types of unlearned anomalous behaviors were present in the sensornet deployment (e.g. an unknown event, a new type of un-modeled fault, or a different undetected malicious data behavior). A false positive is a node that is classified as anomalous but in fact is a normal node. Since applications such as environmental monitoring rely on the fact that sensor readings are temporally and spatially correlated, missing few values from sensor motes producing the false positives does not harm the fidelity of the data and application administrators or data analysts may draw similar conclusions regardless of the missing readings (i.e. a very low false positive rate is acceptable). On the contrary, false negatives are those anomalous nodes that are misclassified as normal. If collected sensor data are visualized at the base station, application administrators can visually inspect the small number of false negatives, model the new behavior, and re-launch the training phase.

Notice that the detector drops a packet if any of the sensing modalities exhibit an anomalous behavior. For instance, if the sensor node incurs an outlier in its temperature readings, the other legitimate light and acoustic readings at the time of the outlier detection are wasted. We understand that this is valuable information and should not be dropped. However, similar to false positives, we believe that the few number of omitted light and acoustic readings are well substituted by other spatially correlated readings from its neighboring nodes. A sensor node may proceed depending on the type of anomaly:

**Fault Detection**

If a node samples faulty readings over $T_p$, the on-line detector running on that node identifies such behavior and sends an alarm signal to the reporting (i.e. sensor collection) module within the same node. In turn, the reporting module suppresses the sample obtained at the
end of the current extraction window $T_c$ (see Figure 6.3(b)). Since some faulty sensors may exhibit temporary faulty characteristics and may return to normal operations (e.g. a wet sensor is dried up), the on-line feature extraction phase continues to capture such characteristics over time. If the sampled readings return to normal, the extracted feature vector should pass the detector and the reporting module keeps forwarding readings up the tree.

**Event Detection**

Assume we have trained the network to detect unauthorized access to a room (i.e. light is turned on suddenly for a short period of time). If such an event happens, the detector identifies the abnormal data behavior caused by the event and sends an alarm signal to the base station. Otherwise, a node can keep silent and will not trigger any alarm.

**Compromise Detection**

As discussed earlier, once a node is compromised, the code, cryptographic keys, and data are all exposed to the adversary. Hence, it is ineffective to rely on the compromised node itself to filter the anomalies as the adversary may completely eliminate the node’s detector. To this end, the detection is applied on parent nodes in the tree every time the parent node receives a new feature vector from its child nodes (Figure 6.3(c)). The detector passes an internal alarm signal to the forwarding module once it detects a falsified reading from one of its child nodes. The forwarding module, in turn, denies forwarding readings sent by the misbehaving child node. Integrity of the sensor data becomes critical if the compromised node is located closer to the base station. In the worst case, the compromised node may intercept every single packet passing through it and alter its readings. In this case, readings from a large number of sensor nodes in the subtree (the compromised node is the root of the
subtree) will be discarded. To avoid missing benign readings, multi paths can be leveraged
if the sensornet employs a multi-path collection/routing protocol.

6.3.8 Re-training / Feature Re-selection

POND can adapt to different types of data-centric anomalies once installed. For in-
stance, network administrators may set up the network to initially detect certain events
(e.g., fire). Later on, it may become desirable to detect other types of events (e.g.,
movement) and possibly filter out outliers in other sensors (e.g., humidity). Each of
these anomalies affects different sensors (e.g., temperature vs. magnetometer and hu-
midity) and therefore it is necessary to re-train the network using a new feature vector
\( \mathbf{f} = \{ f_{1,1}, f_{1,2}, \ldots, f_{1,K}, f_{2,1}, f_{2,2}, \ldots, f_{2,K}, \ldots, f_{N,1}, f_{N,2}, \ldots, f_{N,K} \} \), where \( K \) here is the set
of new modalities (e.g., magnetometer and humidity). It is assumed that the new event or
anomaly is known in advance and can be modeled using one of the anomaly models we
designed in Chapter 4. If this is not the case, then further inspection is considered and a
new anomaly model is designed before the anomaly can be injected into the network. Also
notice that \( N \) can be changed to only cover those useful features. For example, it suffices
to extract the rate of change feature in case of an outlier. Decreasing \( N \), however, does not
necessarily decrease the classifier’s complexity since the ADTree algorithm we discussed
chooses the feature that results in the best split (most useful) at each iteration.

Another scenario where re-training may be required is when new sensors are installed
in the field. Each sensor is usually attached to the mote and motes are re-programmed to
collect readings about the new phenomenon. In order for POND to detect anomalies gener-
ated by the new sensor, the feature space \( \mathbf{f} \) needs to be expanded to cover the new modality
If the feature space remains unaltered, the detection accuracy of the ensemble classifier may drop dramatically. In the evaluation section, we will see that removing the absolute distance feature which is extracted over the light readings introduces a false negative rate of 4% due to inability of detecting clipping faults.

6.4 Aspects of Implementation

As we discuss later in the evaluation section, we have used TinyOS as the underlying operating system running on the sensor nodes. The TinyOS system, libraries, and applications are written in nesC; a simplified variant of the C-language. To generate the online predictor, we have used the JBoost library. A function call passes two references to the generated C predictor, one for an array of pointers containing the feature vector and one representing the scores. Since JBoost supports text attributes in addition to attributes of type double, we can eliminate a large number of the original code space. The code for handling text attributes comprises of dynamic memory allocation of a large hash table of size 1031. Eliminating such code avoids using the unnecessary malloc and realloc functions. While malloc is resource consuming and can cause the sensor node to crash [68], realloc is in fact not supported in nesC. Reducing the code space of the online detector to its minimum and uses only static memory allocation will save us space in the program memory and avoids crashes in data memory.

6.5 Evaluation

To evaluate the effectiveness of our proposed POND framework, we rely on both simulations as well as real-world experimental deployments. In both cases, the target platform
was MicaZ [118] and the sensor board mounted on top of the mote was chosen to be the MTS300. We used three sensors present on the board: (i) temperature, (ii) light, and (iii) acoustic. Moreover, the feature extraction was carried out over only one extraction window ($\lambda = 1$). The latter parameters were adequate to achieve a high detection accuracy in a short period of time.

6.5.1 SNMiner Toolkit

Over the past year, we have developed a tool called SNMiner that enables us to inject anomalies into an existing sensor network deployment, disseminate various parameters, collect/extract feature vectors, and evaluate a set of learning algorithms. Figure 6.4 depicts the building blocks of SNMiner. Rather than relying on publicly accessible datasets, using sensor datasets from earlier deployments [102][105][112], or generating synthetic sensor data, SNMiner models faults and malicious behaviors under the dynamics of a realistic sensor network scenario/setup. We utilized SNMiner to inject anomalous models (similar to those listed in Table 6.1) during both simulation and the real-world experimental deployment. The various mechanisms for injecting anomalies into an existing deployment were
discussed in Section 6.3.1. Although not part of the primary goals of SNMiner, the tool also enables injection of anomalies into sensor datasets obtained from earlier deployments such as the IBRL deployment. In this case, rather than receiving readings from the network, sensor readings are retrieved from a sensor database. To inject faulty readings into a sensor dataset, SNMiner reads the true readings of a selected sensor node over a certain window, transforms the readings based on the desired fault model (see Section 4.2), and then writes the readings back into the database. This is done one reading at a time in order to simulate a real-time injection. Sensor streams are visualized by SNMiner in real-time for early detection of any obvious anomalous behavior (see Figure 6.5(a)). For instance, if one sensor mote reports fully anomalous readings over the initial deployment, it can simply be removed or examined, then recovered prior to training the network or putting it under operation. SNMiner collects feature vectors extracted over the network as discussed in this chapter. Features can also be extracted from an existing sensor database created during the initial deployment or those databases loaded with sensor datasets from previous deployments. Last but not least, few supervised machine learning algorithms (e.g., decision trees, boosting classifiers with different set of base classifiers) are evaluated in real-time. Figure 6.5(b) shows a number of snapshots of the different tabs illustrating the different parameters chosen during the simulation setup discussed next.

6.5.2 Simulation Study

TOSSIM [69] is a discrete event sensor network simulator which simulates an entire, potentially deployable, TinyOS application. Moreover, TOSSIM Live — an extension to the TOSSIM simulator — integrates well with SNMiner. Since simulating sensor readings
is not supported by TOSSIM, we have created a dummy module (DummyC) to replace the modules PhotoC, TempC, and MicC which correspond to the three sensing modalities (temperature, light, and acoustic), respectively. Apparently, using dummy readings does not adequately compare with real sensor values. However, by examining the dataset obtained from an earlier experimental deployment as well as real-world deployments [11], we attempted to generate values that are adequately representative. In this simulation study, we injected a different number of data-centric anomalies into a network of 200 nodes. This was performed using the off-line injection method discussed in Section 6.3.1. There are two injection phases: one for training the classifier and one for testing its effectiveness. The injected anomalies are listed in Table 6.2 and Table 6.3.

We ran the experiment a number of times. In each run, we varied $\beta$ and $T_s$ while maintaining an extraction window of 2 minutes during the training and testing phases. For all runs, we set the number of boosting iterations to 300. There were no instances of
Table 6.2: Injected Anomalies During The Training Phase (Simulation)

<table>
<thead>
<tr>
<th># Nodes</th>
<th>Anomaly Type</th>
<th>Modalities</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Outlier</td>
<td>Temperature</td>
<td>Random $c_k \in C_k = {45, 259, 291, 925, 1009, 1022}$</td>
</tr>
<tr>
<td>2</td>
<td>Clipping</td>
<td>Light</td>
<td>Random $c_k \in {0, 1023}$, over entire window</td>
</tr>
<tr>
<td>3</td>
<td>Event</td>
<td>Light</td>
<td>$o_{29} = 715, o_{30,31} = 272, \sigma = 0, w_{29} = 2, w_{30,31} = 3$</td>
</tr>
<tr>
<td>2</td>
<td>Naive</td>
<td>Temp, Light</td>
<td>$\sigma_{35} = {13, 20}, \sigma_{37} = {18, 23}$</td>
</tr>
<tr>
<td>2</td>
<td>Smart</td>
<td>Light, Mic</td>
<td>$\sigma_{44} = {20, 30}, \sigma_{55} = {25, 35}$</td>
</tr>
</tbody>
</table>

Table 6.3: Injected Anomalies During The Testing Phase (Simulation)

<table>
<thead>
<tr>
<th># Nodes</th>
<th>Anomaly Type</th>
<th>Modalities</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Outlier</td>
<td>Temperature</td>
<td>Random $c_k \in C_k = {45, 259, 291, 925, 1009, 1022}$</td>
</tr>
<tr>
<td>3</td>
<td>Clipping</td>
<td>Light</td>
<td>Random $c_k \in {0, 1023}$, over entire window</td>
</tr>
<tr>
<td>2</td>
<td>Event</td>
<td>Light</td>
<td>$o_{27} = 715, o_{40} = 272, \sigma = 0, w_{27} = 4, w_{40} = 3$</td>
</tr>
<tr>
<td>3</td>
<td>Naive</td>
<td>Temp, Light</td>
<td>$\sigma_{33} = {18, 28}, \sigma_{34} = {22, 18}, \sigma_{41} = {30, 30}$</td>
</tr>
<tr>
<td>3</td>
<td>Smart</td>
<td>Light, Mic</td>
<td>$\sigma_{50} = {18, 25}, \sigma_{52} = {20, 24}, \sigma_{53} = {30, 35}$</td>
</tr>
</tbody>
</table>

Figure 6.6: Detection errors for different values of $\beta$. Three statistical features were extracted (temporal deviation, data range, and highest gradient). Errors are recorded after 300 boosting iterations.

over-fitting. While using large $Q$ ensures the convergence of the boosting algorithm, the boosting algorithm finished earlier during some of the experimental runs as a result of too little weights being placed on the training examples. Figure 6.6 plots the detection error,
false positive rate, and false negative rate for each run. The figure shows that the accuracy of the classifier drops when decreasing the value of $\beta$. This is mainly because the number of samples used to extract the three features (see Section 6.3.2) becomes smaller. The actual detection ratio, however, is still within the range of 98% to 100%. Due to the high packet loss rate, some anomalous samples may even get dropped before reaching the base station. When $\beta$ was set to 20, the classifier could identify all anomalies using a combination of only 6 decision stumps or rules. This shows the advantage of using a combination of simple classifiers (here, decision stumps) rather than a single complex classifier. The final classification label is determined using the resulting ADTree of Figure 6.7. A score is computed by adding up all prediction nodes (blue ellipses) traversed via the true edges of decision nodes (red boxes). If the final score is negative, then the node is classified as anomalous. Finally, we considered the case where one of the sensing modalities was not included in the feature extraction phase. In this case, clipping faults in the light readings were missed during the detection (i.e., testing) phase resulting in a false negative ratio of 4%. This suggests that including more features can increase the classifier accuracy.

As mentioned earlier, there was a large number of lost packets during simulation. If anomaly injection was performed in-network (as done during our real-world deployment),

![ADTree classifier](image-url)
Table 6.4: Injected Anomalies During The Training Phase (Real-World Experiment)

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Anomaly Type</th>
<th>Modalities</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,5</td>
<td>Outlier</td>
<td>Temperature</td>
<td>Random $c_k \in C_k = {45, 259, 291, 925, 1009, 1022}$</td>
</tr>
<tr>
<td>8</td>
<td>Clipping</td>
<td>Light</td>
<td>Random $c_k \in {0, 1023}, w_k = 100$ sec</td>
</tr>
<tr>
<td>13</td>
<td>Spike</td>
<td>Temperature</td>
<td>Following a 3rd degree polynomial regression model with $\beta_0, ..., \beta_3 = {532.8, 0.2397, -0.001264, 4.433e-06}$</td>
</tr>
<tr>
<td>20,21</td>
<td>Smart</td>
<td>Light, Mic</td>
<td>$\sigma_{20} = {20, 30}, \sigma_{21} = {25, 35}$</td>
</tr>
</tbody>
</table>

Table 6.5: Injected Anomalies During The Testing Phase (Real-World Experiment)

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Anomaly Type</th>
<th>Modalities</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Outlier</td>
<td>Temperature</td>
<td>Random $c_k \in C_k = {45, 259, 291, 925, 1009, 1022}$</td>
</tr>
<tr>
<td>11,12</td>
<td>Clipping</td>
<td>Light</td>
<td>Random $c_k \in {0, 1023}, w_{11} = 80$ sec, $w_{12} = 120$</td>
</tr>
<tr>
<td>18</td>
<td>Spike</td>
<td>Temperature</td>
<td>Following a 3rd degree polynomial regression model with $\beta_0, ..., \beta_3 = {532.8, 0.2397, -0.001264, 4.433e-06}$</td>
</tr>
<tr>
<td>19,20</td>
<td>Smart</td>
<td>Light, Mic</td>
<td>$\sigma_{19} = {15, 25}, \sigma_{20} = {10, 24}$</td>
</tr>
</tbody>
</table>

our tests of the detection accuracy should improve since in-network injection ensures all samples (normal and anomalous) will be used during feature extraction.

6.5.3 Real-World Experimental Deployment

We set up an in-door experimental deployment of 22 MicaZ motes randomly scattered over a grid of $6ft \times 4ft$. The motes were initially programmed with the POND application which does not include the detector code. As before, we injected different number of anomalous behaviors into the network during both the training (Table 6.4) and testing (Table 6.5) phases. Here, 10 readings were sampled ($\beta = 10$) at a sampling frequency of 30 seconds. The latter parameters were adequate to achieve a 100% detection accuracy in a short period of time (i.e., 5 minutes). Figure 6.8 shows the detection accuracy of the classifier (Figure 6.9) when extracting all three features over all sensing modalities. It took 7 iterations to eliminate all false negatives. There were no false positives since the first iteration.
6.5.4 Analytical Evaluation

Communication Cost

As discussed in Section 6.3.1, injection of anomaly models may be performed either offline or online. If we choose to perform the injection at the base station and to achieve a rapid detection, we need to instruct the nodes to send their readings faster than the reporting frequency. This incurs an extra communication overhead of $O(\beta \times \lambda \times h)$ per node, where
\( h \) is the number of hops to the base station. Recall that rapid detection is one of the main goals of our proposed framework. If we merely rely on the readings reported at the original rate, the learning cost would be entirely eliminated, but this does not meet the original goal of achieving faster detection. If otherwise the injection is to be performed at each node in the network, the additional communication cost depends on the injection method used. For example, if we solely rely on source routing to deliver the anomaly model to the selected nodes, polling messages are first disseminated into the network to obtain the tree structure. Once the topology information is known, it requires \( O((h + P)h) \) to deliver the model to each selected anomalous node, where \( P \) is the number of model parameters.

During the feature collection phase, every node only needs to communicate its feature vector every \( T_p \) for \( \lambda \) consecutive extraction windows. The feature vector will be forwarded by all intermediate nodes along the path to the base station. Hence, this translates to \( \lambda \times O(hNK) \). This only happens during the training/re-training phase in case the network administrator trains the network to detect measurement faults and/or events. On the other hand, if the sensor network is being deployed in a hostile environment, an additional communication cost of \( \lambda \times O(NK) \) is incurred with every reported sensor reading. This is because after the training phase, a node will continue to transmit its feature vector to its parent.

**Computation Cost**

During the feature extraction phase, computing the feature vectors by itself is a light weight process compared to other previously proposed detection mechanisms. A couple of variables \( V \) is updated at each sample leading to a total of \( \lambda \times O(\beta VNK) \) prior to detection.
In order to check if the node is anomalous, the resulting ADTree is traversed a number of times to compute the final score. The latter process depends on the tree structure. For instance, referring to Figure 6.7, only 6 additions and 5 checks are needed before the node can be determined as either anomalous or normal. As can be seen, the computation complexity of POND during the detection phase is bounded by a small number of checks and additions.

**Power Consumption**

By simply relying on the data sheets of the various sensors and the mote’s hardware components, we can estimate the total energy consumed during both communication and sampling. The total energy consumed during communication can be computed as

\[ E_{\text{total}} = (h - 1) \times b \times V (I_{TX} + I_{RX}) T_X, \]

where \( b \) is the total number of bits to be transmitted, \( V \) is the operating voltage (assuming a fully charged set of AA batteries, \( V = 3 \) Volts), \( I_{TX} \) is the current drawn when transmitting one bit of information, and \( I_{RX} \) is the current drawn per bit during receive mode. \( T_X \) is the time spent during transmission or reception of one bit. The CC2420 radio chip [21] is commonly installed in most of the popular sensor platforms (e.g., MicaZ, TelosB). For this particular chip, \( I_{TX} \) varies from 8.5\( mA \) at a transmission power of \(-25 dBm\) to 17.4\( mA \) at 0\( dBm \), \( I_{RX} \) is 19.7\( mA \), and the data rate is 250kbps (\( T_X = 4 \mu s \)). Using 3 sensing modalities, a TinyOS packet would be of size 24 bytes including header information. Given these values and assuming 6 hops, the total energy spent on communicating one data packet to the base station ranges from \( 324.864 \mu J \) to \( 427.392 \mu J \).

In contrast, the energy consumed when sensing basic environmental phenomena is an
order of magnitude smaller. Power consumption of sensors highly depends on the type and specifications of the sensor device (current draw, data acquisition rate, etc.). For instance, the total amount of power dissipated when sampling a 12-bit value of the Sensirion SHT11 humidity sensor, which is embedded onto the TelosB main board, is 80 µW (at 1 measurement/s) [111]. Given that a 12-bit measurement is obtained within a maximum of 80 ms [111], a node consumes a maximum of 6.4 µJ when sensing one humidity reading.

With the SHT11 sensor, the humidity measurement resolution can be further reduced to 8 bits to enable extreme low power applications. In this case, the total energy consumed when sensing an 8-bit humidity reading will become 1.6 µJ, since the time it takes to obtain an 8-bit measurement is 20 ms at maximum. Assuming $\beta \times \lambda$ is 10, this sums up to 16 µJ. If an anomalous reading is detected using POND, this can use 20 times less energy compared to communicating the anomalous reading to the base station using the lowest transmission power (i.e., $-25 dB$). Recall that sampling the sensors at a higher frequency is only used to achieve rapid detection. It is possible for the node to sample readings at the original rate but this would require $\beta$ samples for the node to detect a possible anomaly.

**Memory Footprints**

MicaZ, one of the very popular commercial sensor platforms, has only 4KBytes of RAM. TelosB, on the other hand, offers 10KBytes of data memory. Therefore, in most cases, we should wisely utilize the RAM to store the least amount of information when possible. Since feature extraction is done on the fly, POND keeps track of a very small number of variables over the extraction window as discussed in Section 6.3.2. If we denote the average number of variables for a specific feature as $V$, then the total memory cost will be
Our entire POND implementation required a total of 2439 bytes of RAM.

In terms of program memory, the detection code fits well into the TelosB program memory of 48KBytes along with other necessary protocols used in a sensor network deployment (DRIP, CTP, and SRP). TelosB is considered one of the most resource constrained devices. MicaZ offers higher program memory of 128KBytes which allows a larger code space but is not essential to the operation of POND. Our entire POND mote application required a total of 29810 bytes of flash.

Table 6.6 provides a complexity comparison of our proposed POND framework against two other popular anomaly detection schemes: (i) an unsupervised clustering-based scheme \([103]\) (ELL), and (ii) the supervised Bayesian-based scheme proposed in \([28]\) (BAYES).

ELL works as follows. Each sensor node stores \(R_w\) observations of dimension \(K\) \((R_w \gg K)\) over a certain time window. Each node then computes the sample mean and the \(K \times K\) covariance matrix of the \(R_w\) observations. The latter computation incurs a complexity of \(O(R_wK^2)\). It then takes the inverse of the covariance matrix—\(O(K^3)\) using Gaussian elimination—to form a hyper-ellipsoid. The resulting hyper-ellipsoidal parameters (i.e., sample mean and inverse covariance matrix) are then used to find the Mahalanobis distance between all observations and the sample mean. If the latter is greater than a certain threshold, the sensor observation at time \(t\) is considered a local anomaly, otherwise, it is locally normal. Computing the Mahalanobis distance by itself requires multiplication of three matrices. After this, a node communicates its elliptical parameters as well as the number of its samples to its parent node. This incurs a communication cost of \(K + (K(K+1)/2)+1\) since the covariance matrix is symmetric. In case of faults or events, this communication cost is replicated after every time window. A parent node merges its children hyper-ellipsoidal
Table 6.6: Comparison of Total Complexities Per Node

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Communication Complexity</th>
<th>Computation Complexity</th>
<th>Memory Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>POND</td>
<td>$\lambda \times O(hNK)$</td>
<td>$\lambda \times O(\beta V NK)$</td>
<td>$O(VNK)$</td>
</tr>
<tr>
<td>POND-HOSTILE</td>
<td>$\lambda \times (O(hNK) + O(RNK))$</td>
<td>$\lambda \times O(\beta V NK)$</td>
<td>$O(VNK)$</td>
</tr>
<tr>
<td>ELL [103]</td>
<td>$O(WK^2)$</td>
<td>$O(RK^2)$</td>
<td>$O(R_w K)$</td>
</tr>
<tr>
<td>BAYES [28]</td>
<td>$O(em^3)$</td>
<td>$O(em^3)$</td>
<td>$O(m^3)$</td>
</tr>
</tbody>
</table>

Table 6.7: Description of Parameters Used In Table 6.6 And Section 6.5.4

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Typical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>number of consecutive extraction windows</td>
<td>1 or 2</td>
</tr>
<tr>
<td>$\beta$</td>
<td>number of samples within one $T_p$</td>
<td>10</td>
</tr>
<tr>
<td>$h$</td>
<td>number of hops to BS</td>
<td>6 (e.g., IBRL [11]: in-door deployment of 54 nodes)</td>
</tr>
<tr>
<td>$P$</td>
<td>number of injected model parameters</td>
<td>1 (outlier in temperature only)</td>
</tr>
<tr>
<td>$N$</td>
<td>number of features</td>
<td>3 (std, max gradient, range)</td>
</tr>
<tr>
<td>$K$</td>
<td>number of sensing modalities</td>
<td>3 (temperature, light, acoustic)</td>
</tr>
<tr>
<td>$V$</td>
<td>avg. number of variables during extraction</td>
<td>(std = 3, max gradient = 2, range = 2)</td>
</tr>
<tr>
<td>$R$</td>
<td>total number of readings</td>
<td>28800 (for $T_c = 30$ seconds, 10-day deployment)</td>
</tr>
<tr>
<td>$W$</td>
<td>total number of windows</td>
<td>10 (1-day window from a 10-day deployment)</td>
</tr>
<tr>
<td>$R_w$</td>
<td>number of readings within window $w$</td>
<td>2880 (1-day window with $T_c = 30$ seconds)</td>
</tr>
<tr>
<td>$m$</td>
<td>number of data ranges in BAYES [28]</td>
<td>5</td>
</tr>
<tr>
<td>$e$</td>
<td>number of epoch in BAYES [28]</td>
<td>2</td>
</tr>
</tbody>
</table>

parameters with its own parameters—adding additional complexity—and communicates the results up the tree. As an additional communication overhead, the gateway node needs to communicate the globally computed hyper-ellipsoid over each detection window back to all the nodes in the network where a node would find global anomalies by computing the Mahalanobis distance once again and comparing it with the threshold. Compared to the communication cost of ELL, POND requires no additional communication during the detection phase which is an order of magnitude savings in terms of communication.

In addition to the higher computation and communication cost incurred by ELL, the total amount of information stored by POND is much smaller than that stored by ELL. ELL requires all sensor readings within a time window (i.e., $R_w$) to be stored in memory and usually $R_w$ is much larger than $K$. Referring to the same example in [103], storing 3000 data points of 5 sensing modalities each requires less than 30100 bytes (assuming the
size of a sensing modality value is 2 bytes) large enough not to fit in the data memory of
the most popular sensor platforms.

BAYES \textsuperscript{[28]} is a naive Bayesian based outlier detection scheme which highly relies on
the fact that the entire observation history of the sensor node is summarized by the previous
reading and the node’s immediate neighbors (parent and child). This assumption holds if
the sensor readings are both spatially and temporally correlated. The scheme uses $m$ data
ranges to predict the missing reading or replace its detected anomalous reading with a more
likely reading within the predicted data range. In order to estimate the probabilities of the
Bayesian model, a total of $(1 + m + 3m^2/2 + m^3/2)$ counters are needed to be stored at
each node. If the phenomena is stationary over space, the learning is performed in-network
whereby each node communicates its local counters to its parent after every specified epoch
and then resets its counters to zero. The parent node, in turn, sums up its children counters
with its own and forwards the resulting counters up the tree. This process continues until
the base station receives the last counter summaries and accumulates them over a number
of epochs $e$. The latter process incurs a communication overhead of $e \times O(m^3)$ for each
node.

The above two schemes are not effective against compromise behaviors since an adver-
sary may entirely eliminate the detection phase. In addition, our scheme detects anomalies
much faster than these schemes as it relies on samples obtained earlier than the reporting
time.
6.6 Limitations and Evasions

This section is mostly addressing the limitations and evasion to our framework in case of a compromised behavior. In identifying events and measurement faults, the only issue is the scalability limited by the size of the TinyOS packet needed to communicate the feature vectors during the training phase to the base station. Suppose that each sensor node has 3 sensing modalities and TinyOS is powering the nodes, each node can typically fit 6 statistical features of 4-byte long (2 features for each modality) in a single TinyOS packet with a 29-byte payload. If the feature vector is longer than a single TinyOS payload, then it has to be divided into two or more packets and hence increasing the communication overhead during the training phase. In addition, during the detection phase, energy consumption increases with the increase of the number of sensing modalities ($\beta > 1$). If the anomaly is caused by a compromised node, however, the packet size limits the amount of communicated features during both the training and detection phases. Moreover, in case of training a network of sensor nodes with a high rate of compromised behaviors/nodes (about 40% of the total nodes), the compromise node ids cannot fit one single disseminated message if injection was done through dissemination.

Furthermore, a compromised sensor node may decide not to communicate its feature vector up the tree. In this case the parent will assume that this child node is misbehaving and can ignore readings received from that node.

In a normal setting, the base station would launch the learning process. Since the code of a compromised node is exposed to the adversary, he/she may leverage the employed configuration dissemination protocol to instruct the network to re-train itself. Thus, we
may only restrict learning to the initial deployment phase to avoid further injections of anomalies. Alternatively, we may reprogram the sensor motes to accept injected messages upon re-training and then reprogram them back using the original operating code. This will again eliminate the injection phase during the online detection, however, it incurs a high communication overhead due to frequent OTA reprogramming.

Finally, a compromised node can communicate a falsified feature during the online detection phase. This will mislead the parent node into thinking that the compromised node (its child) is a legitimate one prior to generating faulty/anomalous readings. To address compromise nodes common in hostile deployments, a simple detector may run on the base station to check the current variations in each node’s consecutive readings. If a deviated behavior is observed, the feature extractor may be performed off-line over the current sensor dataset. The latter assumes that each sensor reading received by the base station every $T_c$ is stored into a sensor database. This, however, will slow down the detection of the falsified readings. Alternatively, we can use a more robust approach based on the same assumption discussed earlier in section 6.3. CTP information can be leveraged to create a global topology view at the base station. A sensor reading is considered anomalous if its spatial distance from its neighbors is higher than a certain threshold. Clearly, the benefits of online detection are entirely vanished when relying on this workaround since the malicious readings are still communicated all the way up the tree. In future work, we plan to come up with a robust mechanism that runs on parent nodes which verifies the integrity of the feature upon receiving the summary from a potentially compromised child node.
6.7 Discussion And Future Directions

This section discusses contributions and stimulate some thoughts about the future usability of the proposed POND framework.

Contributions. The contributions of our work are three folds: (i) the design of an online data-centric anomaly detection mechanism which is fast and highly accurate in identifying measurement faults and events, utilizes very little memory space, and incurs no communication overheads (during the detection phase). Taking actions upon detection is not a critical task in our design. However, we discuss two techniques to act upon the detected flaws in the sensor data by either entirely discarding the faulty readings or raising an alarm in case of a detected event. Once again this depends on the application of the sensornet, if an application administrator is only concerned about an event occurring in a specific region, delivering every data point to the base station creates an unnecessary overhead. This design choice is not unique and can be adjusted to different considerations; (ii) the proposed detection framework can be tailored to sensornets deployed in hostile environments. The major concern in these environments is the trustworthiness of the collected sensor data. If we allow each parent node to examine a feature vector sent by its children, the parent node may take further actions to discard the next reading sent by a misbehaving child node (in terms of the data it sends). ; (iii) we demonstrate the applicability, adaptability, efficiency, and effectiveness of our online anomaly detection framework in real-world scenarios. To the best of our knowledge, our work serves as the first attempt to deploy a practical, rapid, rather effective online data-centric anomaly detection in a real-world sensor network. An earlier attempt to construct such a system [15] suffered from high program memory re-
quirements (see Section 5.1). Furthermore, the system in [15] does not allow re-training of the detector once deployed. In the future, we may also use a public sensornet testbed to further justify the effectiveness of POND.

**Functional Failures and Attacks.** The generality of our framework makes it more flexible and easily expandable. Besides measurement faults, it can be extended to monitor and detect functional faults such as network and node failures (see Section 3.1). This may be achievable by identifying a set of features which capture the behaviors of such failures. It can also address malicious network behaviors and attacks such as sinkholes and wormholes [130]. Studying the behavior of the network under attack may help network administrators determine the appropriate features [76] during the on-line feature extraction phase. For instance the number of received packets from a child node over the extraction window may serve as an indicator of a replay attack. Hence, the ultimate goal is to employ the proposed framework in an existing sensor network deployment to identify a large number of anomalous activities incorporating a set of suitable features into the online detector. As a future work, we are planning to measure the relative importance of each feature (ranking) in identifying the current anomaly and use this to infer the type of anomaly (e.g. if the highest gradient had the highest ranking, it is most probably an outlier or spike fault).

**Distinguishing Among Anomalies.** In this work, we considered that all types of data-centric anomalies involve an anomalous behavior without distinguishing among them. The distinction between events and measurement faults or faults and compromised behaviors is outside the scope of this dissertation. In Section 5.2.3 we formulated the problem of identifying anomalous behaviors as a two-class classification problem in which a node may either be labeled as anomalous or normal. Although this is adequate to achieve the goals
in this dissertation, we believe that using multiple labels brings up an additional benefit: distinguishing among the different types of data-centric anomalies. We are planning to use multi-label classifiers in the future. Moreover, the detector proposed in this chapter drops a packet if any of the sensing modalities exhibit an anomalous behavior (other than an event). For instance, if the sensor node incurs a noise fault in its temperature readings, the other legitimate light and acoustic readings along the same detection window are wasted. We understand that this is valuable information and should not be dropped. However, similar to false positives, we believe that the few number of omitted light and acoustic readings are well substituted by other spatially correlated readings from its neighboring nodes.

**Experimental Evaluation.** In Section 6.5.4 we analytically evaluated the performance of the POND framework in terms of power consumption and communication. In the future, we wish to calculate the actual number of packet transmissions and measure the true power dissipation during each state of the sensor node (i.e., computing, sensing, and communication). The latter leads to a more accurate profiling. We could also rely on simulation and utilize power profilers such as Quanto\[^{31}\] to achieve finer-grained profiling (TOSSIM does not support gathering power measurements).

### 6.8 Applications

POND mainly targets sensor networks deployed for the purpose of monitoring the environment, event-driven deployments, and those sensornets deployed in hostile environments. We discussed a number of such deployments in Chapter 2. In this section, we briefly list two other potential applications that may benefit from our proposed framework.
6.8.1 Business Applications

A good number of business applications, which leverage the newly emerged sensor network technology, could benefit from an anomaly detection scheme like the one we proposed. For example, in [10], the authors measure the quality of goods (fruits, pharmaceuticals) in a temperature controlled supply chain during storage and transport in real-time. Our proposed scheme can be utilized to avoid quality degradation and spoilage by accurately detecting temperature differences at the sensor level and consequently improving the shelf life prediction algorithm.

6.8.2 Health-Care

Like other invaluable data sources, medical data should be protected from any quality degradation whether attempted by faulty health sensors or by user actions [62]. Medical readings can be easily polluted by various artifacts. As an example, correct heart rates cannot be determined in presence of motion artifacts. Unlike less sensitive measurements in different applications, medical measurements can not highly deviate from the expected reading. As another example, in pulse oximetry applications which measure the amount of oxygen concentration in blood, sensor readings cannot deviate higher than 4% from the actual oxygen levels [62]. Our proposed online anomaly detection algorithm may be deployed directly on medical devices to detect artifacts or unexpected deviations and filter them out before collection. This improves the quality of the collected medical data and avoids any false alarms for prompt monitoring of patient’s health.
6.9 Summary

In this chapter, we proposed a supervised online anomaly detection framework that can be adaptable to a number of sensornet deployments: those that enable event-driven applications, deployments for environmental monitoring, and deployments in hostile environments. All deployments share a common challenge: the prevalence of data-centric anomalies. Our framework is able to detect data-centric anomalies rapidly and with low false positives, while at the same time reducing communication overhead and utilizing a small amount of memory space. The practicality and low complexity imposed by ‘online’ feature extraction and detection justify the use of our framework in real-world existing sensor network deployments.
Chapter 7

Conclusions

Existing sensor data acquisition schemes blindly collect raw sensory data without examining its quality. Studying a number of past, existing, and futuristic sensor network deployments in this dissertation have emphasized the need for building a system that ensures the fidelity, integrity, and/or trustworthiness of the collected sensor data. Usually, there are two design choices whenever deciding to build such a system. The first would come up with a detection mechanism which runs at the centralized base station and may identify abnormal readings and filter them accordingly. Given the fact that sensor nodes in a sensor network deployment are highly resource-constrained, transmitting anomalous readings all the way to the base station consumes the node’s scarcest resource (i.e., energy) at an order of magnitude greater than simply discarding it. To this end, online detection mechanisms that run directly at the sensor level are more preferable especially if the number of anomalous readings is non-negligible.

This dissertation proposed both a centralized and an online data-centric anomaly detection frameworks which were capable of identifying measurement faults as well as compromise data behaviors. These frameworks are rooted in machine learning and require a highly expressive ground truth. In the dissertation, we also proposed and designed a number of
anomaly models that proved to be significantly representative of the anomalous behaviors in a real-world setting. Using a tool we develop at our lab, we used such models to inject anomalous behaviors into a sensor network deployment in real-time, extract simple statistical features, and evaluate the accuracy of a number of supervised learners. We have shown that, in some cases, both frameworks identified data-centric anomalies with negligible or non-existent false positives and false negatives. Finally, to prove the practicality of the online framework, it was entirely implemented using the TinyOS programming language which operates a large number of today’s sensor motes.
Bibliography

[1] *Cosm: Connect to your world.*


on Quality of service & security in wireless and mobile networks (Q2SWinet), pages 16–23, 2005.


[58] Mohammad Maifi Hasan Khan, Hieu K. Le, Michael LeMay, Parya Moinzadeh, Lili Wang, Yong Yang, Dong K. Noh, Tarek Abdelzaher, Carl A. Gunter, Jiawei Han, and Xin Jin. Diagnostic powertracing for sensor node failure analysis. In *Proceedings of


[76] Chong Eik Loo, Mun Yong Ng, Christopher Leckie, and Marimuthu Palaniswami. Intrusion detection for routing attacks in sensor networks. *International Journal of*


[128] Tran Van Phuong, Le Xuan Hung, Seong Jin Cho, Young-Koo Lee, and Sungyoung Lee. An anomaly detection algorithm for detecting attacks in wireless sensor net-


