A Learning-based Approach to Exploiting Sensing Diversity in Performance Critical Sensor Networks

Matthew Alexander Keally

College of William & Mary - Arts & Sciences

Follow this and additional works at: https://scholarworks.wm.edu/etd
Part of the Computer Sciences Commons

Recommended Citation

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.
A Learning-based Approach to Exploiting Sensing Diversity in Performance Critical Sensor Networks

Matthew Alexander Keally
Virginia Beach, Virginia

Master of Science, The College of William and Mary, 2008
Bachelor of Science, The College of William and Mary, 2006

A Dissertation presented to the Graduate Faculty of the College of William and Mary in Candidacy for the Degree of Doctor of Philosophy

Department of Computer Science

The College of William and Mary
May 2012
This Dissertation is submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Matthew Alexander Keally

Approved by the Committee, March 2012

Committee Chair
Assistant Professor Gang Zhou, Computer Science
The College of William and Mary

Weizhen Mao

Associate Professor Weizhen Mao, Computer Science
The College of William and Mary

Associate Professor Haining Wang, Computer Science
The College of William and Mary

Professor Virginia Torczon, Computer Science
The College of William and Mary

Assistant Professor Guoliang Xing, Computer Science and Engineering
Michigan State University
COMPLIANCE PAGE

Research approved by

Protection of Human Subjects Committee

Protocol number(s): PHSC-2010-11-28-7026-pdkieffaber

Date(s) of approval: 2011-01-05
Wireless sensor networks for human health monitoring, military surveillance, and disaster warning all have stringent accuracy requirements for detecting and classifying events while maximizing system lifetime. To meet high accuracy requirements and maximize system lifetime, we must address sensing diversity: sensing capability differences among both heterogeneous and homogeneous sensors in a specific deployment. Existing approaches either ignore sensing diversity entirely and assume all sensors have similar capabilities or attempt to overcome sensing diversity through calibration. Instead, we use machine learning to take advantage of sensing differences among heterogeneous sensors to provide high accuracy and energy savings for performance critical applications.

In this dissertation, we provide five major contributions that exploit the nuances of specific sensor deployments to increase application performance. First, we demonstrate that by using machine learning for event detection, we can explore the sensing capability of a specific deployment and use only the most capable sensors to meet user accuracy requirements. Second, we expand our diversity exploiting approach to detect multiple events using a distributed manner. Third, we address sensing diversity in body sensor networks, providing a practical, user friendly solution for activity recognition. Fourth, we further increase accuracy and energy savings in body sensor networks by sharing sensing resources among neighboring body sensor networks. Lastly, we provide a learning-based approach for forwarding event detection decisions to data sinks in an environment with mobile sensor nodes.
# Table of Contents

Dedication vii

Acknowledgments viii

List of Tables ix

List of Figures x

1 Introduction 2

1.1 Problem Statements 3

1.1.1 Exploring and Quantifying Sensing Diversity 4

1.1.2 Sensing Diversity in Distributed Sensor Networks 5

1.1.3 Sensing Diversity in Body Sensor Networks 7

1.2 Contributions 9

1.3 Dissertation Organization 16

2 Related Work 17

2.1 Exploring Sensing Diversity and Distributed Sensor Networks 17

2.1.1 Diversity Awareness and Meeting User Requirements 18
2.1.2 Data Forwarding in Mobile Environments ........................................ 19

2.2 Sensing Diversity and Body Sensor Networks ..................................... 20
  2.2.1 Providing A Practical Solution ..................................................... 20
  2.2.2 Inter-BSN Resource Sharing ...................................................... 22

3 Watchdog .............................................................................................. 24
  3.1 Motivation ......................................................................................... 25
  3.2 Watchdog Design ............................................................................. 28
    3.2.1 Local Aggregation ...................................................................... 30
    3.2.2 Machine Learning Exploration .................................................. 32
    3.2.3 Cluster Generation .................................................................... 35
    3.2.4 Sentinel and Reinforcement Selection ........................................ 38
    3.2.5 Runtime Event Detection ........................................................... 39
    3.2.6 Online Retraining ...................................................................... 42
  3.3 Evaluation .......................................................................................... 43
    3.3.1 Exploring Detection Capability and Meeting Requirements .......... 45
    3.3.2 Comparison with V-SAM ............................................................ 47
    3.3.3 Comparison with a Modality-Specific Sensing Model .................. 50
    3.3.4 Machine Learning Comparison ................................................... 52
    3.3.5 Online Retraining ...................................................................... 55
    3.3.6 Transmission Energy Savings ...................................................... 58
  3.4 Conclusion ......................................................................................... 59

4 Wolfpack ............................................................................................... 60
4.1 Exploring and Exploiting Sensing Diversity .............................................. 62
  4.1.1 Identifying Sensing Diversity ................................................................. 63
  4.1.2 Impact of Diversity on Collaboration and Accuracy ................................. 64
    4.1.2.1 On-Demand Collaboration ................................................................. 65
    4.1.2.2 Collaboration Accuracy and Complexity of Learning ....................... 67
  4.2 Diversity-Exploiting Confident Coverage ................................................ 68
  4.3 Wolfpack Framework Design ...................................................................... 70
    4.3.1 Distributed Diversity-Aware Clustering .................................................. 71
    4.3.2 Runtime and Adaptive Coverage ............................................................. 76
  4.4 Evaluation ................................................................................................. 77
    4.4.1 Meeting User Requirements ..................................................................... 78
    4.4.2 Exploiting Sensing Diversity ................................................................. 80
    4.4.3 Adaptive Coverage .................................................................................. 81
    4.4.4 Active Nodes and Energy Usage ............................................................. 82
  4.5 Conclusion and Future Work ....................................................................... 84

5 Practical Body Networking .............................................................................. 86
  5.1 System Overview and Architecture ............................................................. 88
    5.1.1 Application Requirements ....................................................................... 88
    5.1.2 PBN Hardware and Application ............................................................. 90
    5.1.2.1 Unification of Android and TinyOS ..................................................... 90
    5.1.2.2 Android App ...................................................................................... 92
    5.1.3 PBN Architecture .................................................................................. 94
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3</td>
<td>Neighbor Management</td>
<td>138</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Proximity Detection and Duration Prediction</td>
<td>138</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Cost-Benefit Analysis</td>
<td>140</td>
</tr>
<tr>
<td>6.4</td>
<td>Sharing-Aware Classification</td>
<td>142</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Sensor Selection Motivation</td>
<td>144</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Collaborative Sensor Selection</td>
<td>147</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Classifier Sharing and Duty Cycling</td>
<td>149</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Empirical Sharing Cost Model</td>
<td>150</td>
</tr>
<tr>
<td>6.5</td>
<td>Evaluation</td>
<td>154</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Proximity and Sharing</td>
<td>154</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Accuracy Improvement</td>
<td>156</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Sharing Costs and Benefits</td>
<td>158</td>
</tr>
<tr>
<td>6.5.4</td>
<td>Energy Savings</td>
<td>159</td>
</tr>
<tr>
<td>6.5.5</td>
<td>CPU and Memory Usage</td>
<td>162</td>
</tr>
<tr>
<td>6.6</td>
<td>Conclusion and Future Work</td>
<td>162</td>
</tr>
<tr>
<td>7</td>
<td>Sidewinder</td>
<td>164</td>
</tr>
<tr>
<td>7.1</td>
<td>Motivation</td>
<td>166</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Impact of Radio Ranges on Topology Changes</td>
<td>166</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Impact of Sink Mobility on Geographic Forwarding</td>
<td>168</td>
</tr>
<tr>
<td>7.2</td>
<td>Sidewinder Overview</td>
<td>170</td>
</tr>
<tr>
<td>7.3</td>
<td>Detailed Sidewinder Design</td>
<td>171</td>
</tr>
<tr>
<td>7.3.1</td>
<td>SMC Prediction Concept</td>
<td>172</td>
</tr>
</tbody>
</table>
7.3.2 SMC adapted to Wireless Sensor Networks ............................... 173
7.3.3 Mobility Monitor .............................................................. 174
7.3.4 SMC Initialization ............................................................. 175
7.3.5 SMC Prediction, Filtering, and Resampling .............................. 179
7.3.6 Adaptive Update ............................................................... 181
7.4 Evaluation ........................................................................... 183
7.5 Conclusion .......................................................................... 186

8 Conclusion and Future Work ...................................................... 188

8.1 Contributions Revisited ......................................................... 188
8.2 Future Work ....................................................................... 193

Bibliography .............................................................................. 197

Vita ......................................................................................... 211
To my wife, Sarah.
ACKNOWLEDGMENTS

This dissertation would not have been possible without the help of many kind, talented, and hardworking people. I truly am standing on the shoulders of giants. I would first like to thank my advisor, Gang Zhou for all of his insight and guidance over the last five years. His passion and selfless dedication to his students makes working with him a pleasure.

Many thanks to my committee members, Haining Wang, Weizhen Mao, Virginia Torczon, and Guoliang Xing for their valuable comments on this dissertation. I am especially grateful to Professor Mao for introducing me to my advisor, Gang Zhou, and giving me the initial encouragement that led me down the road towards this dissertation.

I am also grateful to work with many outstanding colleagues and collaborators. Specifically, I would like to thank Guoliang Xing and Jianxin Wu for their ever present help and feedback. It has been great to see so many students join our research group and provide valuable commentary and discussion. Thanks to all of them for their assistance and insight: Nancy Carter, Kevin Li, Yantao Li, David Nguyen, Ge Peng, Andy Pyles, Xin Qi, Zhen Ren, George Simmons, and Lingfei Wu. The departmental staff, notably Vanessa Godwin and Jacquelyn Johnson, has been especially helpful by guiding me towards graduation.

Lastly, I would like to thank my family for their support and encouragement. I cannot begin to describe the love, patience, and understanding my wife, Sarah, has provided over the years. I am also very grateful to my parents and sister for their encouragement and words of wisdom; their examples have always inspired me.
## List of Tables

3.1 Adapting detection capability with reinforcements. ........................................ 52

4.1 Cluster vs. individual sensor performance. .................................................... 67

4.2 Overall accuracy for Wolfpack and V-SAM. .................................................. 78

4.3 Sensor, modality, and node makeup per cluster. ............................................ 80

4.4 Active nodes and energy usage for Wolfpack and V-SAM. .............................. 83

5.1 PBN Classification Categories. ........................................................................ 88

5.2 PBN Deployment Configuration. ....................................................................... 97

5.3 PBN CPU, memory, and power benchmarks. ................................................... 121

6.1 Deployment Configuration. ................................................................................ 131

6.2 Remora Power Consumption (mW). ................................................................. 151

6.3 Total Proximity and Sharing Duration. ............................................................. 155

6.4 Remora CPU and memory benchmarks. ......................................................... 162
List of Figures

3.1 Event detection performance with vehicle trace data. ...................... 26
3.2 Watchdog design. .............................................................. 29
3.3 Transmission threshold. ....................................................... 31
3.4 Event Probability breakdown. ................................................. 37
3.5 Runtime detection timeline. ................................................... 41
3.6 Cluster training and runtime detection. ...................................... 46
3.7 Watchdog and V-SAM comparison. ........................................... 48
3.8 Watchdog and modality-specific sensing model comparison. .............. 50
3.9 Multiple location performance. .............................................. 53
3.10 Energy usage. ................................................................. 54
3.11 Reinforcement usage and Online Retraining. ............................... 56
3.12 Energy use and accuracy for different transmission threshold values. .. 58

4.1 Sensing range differences in Wisconsin deployment. ....................... 63
4.2 Nearest Centroid .............................................................. 66
4.3 Fisher's Linear Discriminant ................................................. 66
4.4 Distributed clustering. ....................................................... 75
4.5 Location accuracy CDF. .................................................. 79
4.6 Cluster membership CDF. .................................................. 80
4.7 Location updates and accuracy during runtime. ....................... 81
4.8 Updates per location. ..................................................... 82
4.9 Number and accuracy of update types. ................................ 82
4.10 CDF of energy usage per detection location for Wolfpack and V-SAM. ... 84

5.1 PBN activity status view. ................................................... 92
5.2 Ground truth logging. ..................................................... 92
5.3 PBN System Architecture. ................................................ 94
5.4 Subject 1. ................................................................. 97
5.5 Subject 2. ................................................................. 97
5.6 Training and runtime accuracy for 1-300 AdaBoost iterations. .... 101
5.7 Average accuracy for 10-100 random training observations. ....... 101
5.8 K-L divergence for each sensor and each activity. .................... 102
5.9 Sensor K-L divergence and training accuracy. ....................... 103
5.10 Raw data correlation for Subject 1. ................................... 109
5.11 Decision correlation between individual sensors and sensor clusters. ... 110
5.12 Runtime performance for multiple subjects and classification categories. ... 114
5.13 PBN decision and ground truth timeline for Subject 1. ............... 115
5.14 AdaBoost sensor weights per activity. ................................ 116
5.15 PBN retraining instances and runtime accuracy. ..................... 117
5.16 Activity training data balance restriction. ............................ 118
5.17 Retraining instances and runtime accuracy. ........................................ 118
5.18 Runtime timeline with accuracy and retraining instances. .................. 119
5.19 Sensor selection comparison with online training. .......................... 119
5.20 Power Consumption Evaluation. ....................................................... 121

6.1 Sharing Hierarchy. ..................................................................... 128
6.2 BSN Configuration. ................................................................. 130
6.3 Runtime accuracy for shared and individual BSN configurations. ....... 132
6.4 Accuracy for different combinations of on-body sensors. ................... 132
6.5 Randomly selected sensor clusters. ............................................. 133
6.6 Remora Architecture. ................................................................. 134
6.7 Activity status. ..................................................................... 137
6.8 Shared resources. ..................................................................... 137
6.9 Map view of individual and shared activities. ................................. 137
6.10 Subject 1: Individual sensor accuracy. ....................................... 145
6.11 Decision correlation and accuracy improvement. ........................... 146
6.12 Remora power profile. ............................................................... 151
6.13 Power Meter Setup. ................................................................. 151
6.14 Phone training time: sensor classifiers and Collaborative Sensor Selection 152
6.15 Proximity duration needed for sharing to benefit. ............................ 153
6.16 Runtime accuracy for each subject. ............................................. 156
6.17 Subject 1 accuracy. ................................................................. 156
6.18 Subject 4 accuracy. ................................................................. 156
A Learning-based Approach to Exploiting Sensing Diversity in Performance Critical Sensor Networks
Chapter 1

Introduction

The increasing pervasiveness of wireless sensor networks, embedded systems, and mobile phones brings both new possibilities and new challenges to wireless research. Such proliferation has paved the way for a large number of performance critical applications which use mobile devices, such as security enforcement and military surveillance; smart healthcare and assisted living; participatory sensing and social networking; and natural and physical hazard detection (volcanoes, hurricanes, and fires). Performance critical applications such as these demand stringent performance requirements in terms of high accuracy and long system lifetimes. Failure to meet these requirements can lead to undesirable or even catastrophic consequences.

To meet requirements for performance critical applications, we must address the *sensing diversity* present among all sensors in a performance critical deployment. Sensing diversity encompasses the sensing capability differences between sensors of the same modality as well as among different modalities. The causes of sensing diversity can be linked to the in-situ reality of a specific deployment [56] as well as hardware differences [117], especially in the
case of cheap off-the-shelf motes.

Many works ignore sensing diversity entirely and assume all sensors have similar sensing capabilities [132] [133]. Other works attempt to overcome sensing diversity by correcting for the differences in readings from different sensors [117]. Instead, we take advantage of such sensing differences to provide both high accuracy and energy savings. Through the use of machine learning, we are able to capture the sensing capability of each sensor deployment and use the learned sensing capability to meet performance critical application requirements.

1.1 Problem Statements

In this dissertation, we address three significant challenges in meeting performance critical application requirements through exploitation of sensing diversity. First, using machine learning, we must be able to fully explore and quantify the detection capability of a specific sensor deployment. Second, when applying learned sensing capability to distributed sensor networks, we must provide a lightweight and generic approach which allows nodes to easily collaborate and choose the most accurate and energy efficient sensors. A routing solution is also required for mobile wireless sensor networks to forward detection and classification decisions to an end user with minimal packet loss. Third, when applying sensing diversity for personal sensing applications in body sensor networks, we must provide a practical solution which is user friendly and accurate as well as adaptive to the body sensor network dynamics that arise from changing physical location, body movement, and background noise. To provide additional performance improvements over individual body sensor networks, BSNs
can exploit physical proximity and share sensing resources with neighbouring BSNs to achieve both accuracy and energy gains.

1.1.1 Exploring and Quantifying Sensing Diversity

Existing approaches to sensing, event detection, and classification ignore the sensing capability differences among sensors in a deployment. Many existing works ignore sensing diversity and assume that sensors in real deployments have perfect disc patterns [81] or follow a signal attenuation model [136]. Other works assume that all sensors exhibit the same sensing pattern [132] [133], however none of these sensing assumptions hold up in reality [56]. Another work [117] attempts to correct for such sensing differences among sensors with calibration, but this approach still relies on a signal attenuation model. To address these concerns, the following challenges arise: learning sensing diversity, exploring the sensing capability of a sensor deployment, and determining when sensor collaboration is needed.

**Learning Sensing Diversity.** Machine learning techniques, many of which can be utilized to learn the sensing capabilities of a specific deployment, greatly differ in terms of accuracy and complexity. A learning method for energy constrained sensors and mobile devices should provide enough accuracy to meet application requirements but still require low computation and communication overhead.

**Exploring Detection Capability of a Sensor Deployment.** To meet performance critical application requirements in the face of sensing diversity, we must find and use the most energy efficient sensor clusters. Previous work [47] demonstrates that clustering sensors can, in many cases, improve detection accuracy over individual sensors. We must be
able to explore the detection capability of a specific deployment through clustering sensors and choose the right sensor clusters to save energy and meet application requirements.

**On-demand Sensor Collaboration.** Clustering sensors, however, may not always improve sensing performance. In many deployments, individual sensors are sufficient to provide high accuracy, and hence collaboration is not needed. Through learned sensing diversity, it is important to determine when single sensors are sufficient and when collaboration is necessary. When sensor collaboration is needed, it is also important to determine the right sensors to collaborate, for such careful collaboration can save valuable computation and communication resources.

### 1.1.2 Sensing Diversity in Distributed Sensor Networks

Distributed sensor networks are widely used for event detection or classification for performance critical applications in military surveillance [47], environmental and wildlife monitoring [91] [30], as well as vehicle tracking [28]. Sensing diversity is extremely apparent in deployments for such applications, for the deployments often use large numbers of nodes with heterogeneous sensing modalities in harsh and dynamic environments. To meet application requirements in distributed sensor networks, several challenges arise:

**A Generic Solution.** Since distributed sensor networks are used for a wide variety of applications, a generic solution is needed that can easily work with such applications, sensor modalities, and machine learning methods. Since many real deployments use multiple modalities, using different event detection solutions for each modality can be difficult. We must be able to perform efficient collaboration among heterogeneous sensors which works in a generic context.
Efficient Adaptation to Runtime Dynamics. A small energy efficient cluster of sensors may be sufficient to meet user requirements most of the time. Sometimes, however, more detection capability may be required, prompting collaboration between lower power and higher power clusters to meet user requirements and save energy. Clusters and machine learning detection models may also need to be updated during runtime.

Distributed and Online Diversity Exploitation. Distributed learning and exploitation of sensing diversity allows for decreased bandwidth and energy usage as well as increased scalability. Furthermore, a distributed scheme allows for more efficient adaptation to environmental dynamics, for only portions of the network that do not meet user requirements need to be updated during runtime to ensure these requirements are met.

Decision Routing in Mobile Deployments. Many deployments for performance critical applications exhibit node mobility, such as sensor deployments for Micro Air Vehicles [3], bikes [32], vehicles [33] [55], and animals [126] [82]. Detection decisions produced on mobile sensor nodes must be routed to an end user using an approach that can handle volatile topology changes with low packet loss.

Existing approaches for event detection do not provide a holistic solution with respect to addressing these challenges. Some approaches do not work in heterogeneous sensor deployments and do not provide a generic solution, such as data fusion-based modality-specific sensing models [19] [136]. Other sensing models [102] [122] ignore different sensing modalities as well as detection and classification methods by using abstract fidelity functions. None of these works adapt to environmental dynamics in order to meet application requirements. Furthermore, routing approaches for both traditional wireless sensor networks [139] [129] [70] [119] [32] and mobile ad hoc networks [97] [38] [59] [72] [10] do not address the volatile
topology changes present when using low power radios.

1.1.3 Sensing Diversity in Body Sensor Networks

The sensing power of on-body wireless sensors combined with the additional sensing power, computational ability, and user interface of a smartphone makes body sensor networks (BSNs) an ideal platform for performance critical personal sensing applications. Such applications include activity recognition for assisted living [130], physical fitness assessment [2], and patient monitoring [83] [20]. A physician may administer BSNs for retirement community residents [130] [103] to detect depression, ensuring proper eating, social activity, and exercise. Similarly, a university sports team coach may deploy BSNs on his or her athletes to ensure optimal performance [6]. The BSN worn by each athlete can not only measure athletic performance but also detect daily living habits that may be detrimental, such as excessive social activity. Exploiting the sensing diversity present in BSNs presents the following challenges, especially in applications for daily activity recognition:

**A Practical Solution.** We envision a practical solution for BSN activity recognition which is entirely portable, under direct control of the user, computationally lightweight, and accurate. For personal sensing applications, the hardware and software platform must be user friendly and easily configurable with adequate realtime feedback. Classification must be accurate, handling both easy and difficult to classify activities as well as noisy data and environmental dynamics. Also, since mobile hardware is often constrained in terms of computation power and energy, classification must be performed efficiently by finding and utilizing only the most helpful sensors.

**Adaptation to BSN Dynamics.** While static sensor networks are challenged to
handle environmental dynamics, such externalities are amplified in body sensor networks. Compared to static sensor networks, body sensor network (BSN) dynamics include the changing geographical location of the user, user biomechanics and variable sensor orientation, as well as background noise. During runtime, these factors may significantly affect sensor and system performance, triggering online training as well as ground truth labeling from the user. Because labeling sensor data can be invasive to the user, the system must have a reduced reliance on ground truth.

**Inter-BSN Resource Sharing.** In a BSN application for assisted living or physical fitness assessment of a sports team, users spend a significant amount of time with one another while performing the same activities. We can exploit this physical proximity by sharing resources among neighboring BSNs to increase accuracy and energy savings. However, to achieve such performance increases, we must address three major issues: 1) sharing only when the energy benefit outweighs the cost, 2) finding and utilizing the shared resources which produce the best combination of accuracy and energy savings, and 3) providing a classification approach which easily adapts to the dynamics of available neighbors.

While there is significant existing work in the mobile activity recognition domain, no approach addresses all of the above challenges. Some approaches [39] [83] [138] provide multiple on-body sensor nodes but do not provide a portable interface, such as a mobile phone, for the end user to control sampling, configure hardware, or receive real time feedback. More works [86] [87] rely on specific sensing models for each sensor modality, making sensor collaboration difficult. Other works [125] [104] do not provide online training for adaptation to BSN dynamics. Still more works [89] [90] rely on computationally expensive learning algorithms and a back end server, especially those that share resources among users.
One effort [89] shares classifiers and classification results among neighbors but the energy costs and benefits are not fully addressed.

1.2 Contributions

The overall result of this dissertation is significant improvements in accuracy and energy savings for performance critical sensor networks. These improvements are realized through learning and exploiting sensing diversity in each specific sensor deployment. We present five main contributions towards meeting the challenges of exploiting sensing diversity for performance critical applications:

Event detection at a single location. We first focus on meeting user or application accuracy requirements for sensor network-based event detection at a single location, such as detecting vehicles at a bridge crossing or intersection. When an event detection framework meets user accuracy requirements in terms of false positive and false negative rate, we say it is confident. In order to meet user accuracy requirements and make confident event detection decisions, we demonstrate that an event detection framework must be able to capture the sensing capability of different sensors and sensor clusters. Consequently, we propose Watchdog, an event detection framework which explores the detection capability of a specific deployment and chooses the right sensors to meet accuracy requirements. We investigate several machine learning methods that our generic approach can use for event detection in heterogeneous sensor deployments. To adapt to environmental dynamics, Watchdog uses an energy efficient sentinel sensor cluster to make easy event detection decisions. When the sentinel cluster cannot make a confident decision, a more capable reinforcement sensor
cluster ensures the user requirements are met. When persistent changes in the environment are detected which significantly impact meeting user accuracy requirements, we update the detection models during runtime and form new clusters. Our main contributions are:

- With trace data from a vehicle detection application, we show the drawbacks of existing solutions and motivate the need for a confident event detection framework.

- We propose Watchdog, a generic event detection framework which clusters the right sensors to enforce user-defined event detection accuracy during runtime.

- Watchdog efficiently adapts to environmental dynamics by requesting only the sensing capability needed to meet user requirements and updating clusters when user requirements can no longer be met.

- We evaluate Watchdog in two scenarios: a vehicle detection application using trace data and a building traffic monitoring application using IRIS motes. Watchdog can always meet user-specified detection accuracy with reduced energy usage, while in many cases existing solutions cannot.

**Distributed Event Detection.** Next, we capture and explore sensing diversity for distributed sensor networks and event detection at multiple locations, such as detecting vehicles at many locations along a road. We show that sensing capabilities greatly differ among sensors in a real deployment and identify when sensor collaboration is needed. When collaboration is needed, we show that arbitrary sensor collaboration often fails to meet user accuracy requirements, not to mention a joint consideration of accuracy and energy efficiency. We explore different machine learning techniques for on-demand collaboration.
and identify one appropriate for energy and computationally constrained sensor networks. These results provide key insights into protocol design for collaborative sensing.

We are among the first to take advantage of explored sensing diversity to provide sensing confidence and apply this approach for confident sensing coverage, such as in a fixed sensor network deployment for vehicle detection. We formally define and theoretically analyze our confident coverage problem when diversity is explored. We show that a specific case of our diversity-exploiting confident coverage problem is NP-hard and propose Wolfpack, a distributed event detection framework that exploits sensing diversity for use in practical deployments. With machine learning, Wolfpack determines only the sensing capability needed to meet user detection requirements and save energy, only collaborating sensors when individual sensors are not accurate enough. During runtime, Wolfpack adapts its detection capability to adjust for environmental changes that cause a drop in accuracy and run the risk of not meeting the user requirements. Our main contributions are:

- We explore the fundamental challenges in addressing sensing diversity and its impact on collaboration for confident sensing using two different machine learning techniques.

- Through theoretical analysis and our practical Wolfpack design, we exploit sensing diversity to provide sensing confidence and apply it to sensing coverage.

- Our evaluation in a vehicle detection application demonstrates that Wolfpack achieves confident coverage for 30% more locations while using 20% less energy than a state of the art approach.

Practical Activity Recognition for Body Sensor Networks. Towards addressing the challenges of body sensor network-based activity recognition in the face of sensing
diversity, we propose PBN: Practical Body Networking. PBN consolidates the flexibility and sensing capability of TinyOS-based motes with the additional sensing power, computational resources, and user-friendly interface of an Android smartphone. Our solution can also be extended beyond TinyOS motes to combine Android with a wide variety of USB devices and wireless sensors. Through the use of ensemble learning, which automates parameter tuning for BSN dynamics, PBN provides a capable, yet lightweight activity recognition solution that does not require a backend server.

With PBN, we provide an online training solution which detects when retraining is needed by analyzing the information divergence between training and runtime data distributions and integrating this analysis with the ensemble classifier. In this way, PBN determines when retraining is needed without the need to request ground truth from the user. Furthermore, we investigate the properties of sensors and sensor data to identify sensors which are accurate and have diverse classification results. From this analysis, we are able to prevent the ensemble classifier from needlessly consuming computational overhead by using redundant sensors in the online training process. Our main contributions are:

- We combine the sensing capability of on-body TinyOS-based motes with the sensors, computational power, portability, and user interface of an Android smartphone.

- An activity recognition approach appropriate for low-power wireless motes and mobile phones that does not rely on a backend server. Our approach handles BSN dynamics without sophisticated parameter tuning and also accurately classifies difficult tasks.

- We provide retraining detection without requesting ground truth from the user, reducing the invasiveness of the system.
• We reduce online training costs by detecting redundant sensors and excluding them from the ensemble classifier.

• With two weeks of data from two subjects, we demonstrate that we can detect even the most difficult activities with nearly 90% accuracy. We identify 40% of sensors as redundant, excluding them from online training.

**Inter-Body Sensor Network Resource Sharing.** In body sensor networks for activity classification, sensing diversity can be further exploited by sharing sensors and classifiers among neighbors in physical proximity to one another. We show through an initial experiment how sharing sensors among neighboring BSNs can increase activity classification accuracy and save sensor energy by using fewer sensors. The insights gained from this experiment motivate the design of our system, Remora. Remora is an opportunistic resource sharing approach which improves classification accuracy and extends system lifetime among BSNs in proximity to one another. With Remora, we first determine the costs and benefits of sharing: we determine energy overhead as well as the proximity duration needed for the sharing energy benefit to outweigh the energy costs. Next, we provide a sharing-aware classification approach which uses an ensemble classifier that efficiently adapts to changes in neighbor and sensor availability. This approach allows sharing BSNs to jointly select sensors to maximize training accuracy and use as few sensors as possible to save sensor energy. To save phone energy, sharing BSNs only use one active classifier per time period. Our main contributions are:

• We analyze the overhead of sharing sensors and classifiers with a time and energy model, only sharing when neighboring BSNs will be together long enough for sharing
to benefit.

- We provide an efficient method to share sensors and classifiers among neighboring BSNs. A collaborative approach allows neighbors to share only the most accurate sensors and duty cycle classifiers to save phone energy.

- With two weeks of evaluation from six subjects, in comparison with using only individual BSN resources, Remora can increase activity classification accuracy by nearly 30% and extend battery lifetime by over 65%.

Predictive Data Forwarding in Mobile Environments. To address the challenge of routing detection and classification decisions in mobile wireless sensor networks, we provide a data forwarding protocol which learns and exploits node movements to make accurate routing decisions. We perform a quantitative evaluation of traditional mobile ad hoc and wireless sensor routing protocols and determine that increased network dynamics cause the performance of these protocols to degrade significantly. For these reasons, we propose Sidewinder, a routing protocol which predicts node movements when making routing decisions. Using Sequential Monte Carlo prediction, which is adapted for resource-constrained sensor networks, data packets are guided towards a sink node with increasing accuracy as packets approach the sink. Such accuracy is extremely important in meeting user requirements for performance critical applications. Different from conventional sensor network routing protocols, Sidewinder continuously predicts sink and intermediate node locations based on distributed knowledge of sink and neighbor mobility in a multi-hop routing process. Our main contributions towards predictive routing are:

- We demonstrate through quantitative evaluation that traditional mobile ad hoc and
sensor networking protocols cannot handle the excessive topology changes present in mobile deployments.

- We propose Sidewinder, a predictive data forwarding protocol for mobile wireless sensor networks which predicts node movements at each hop to accurately route data from source to sink with low packet loss.

- To predict node movements, we use Sequential Monte Carlo (SMC) theory and apply it in a low overhead manner appropriate for distributed sensor networks.

- We implement Sidewinder in nesC [41] and evaluate it in TOSSIM [98] to demonstrate that Sidewinder significantly outperforms existing solutions for in-situ data collection under intensive topology changes in mobile sensor networks.

Techniques presented in this dissertation will significantly advance the state of the art in providing increased accuracy and energy savings for performance critical applications. Our exploration of sensing diversity will allow a wide range of deployment configurations with different hardware to provide sensing confidence, including both distributed and body sensor network deployments. Furthermore, our development of a data forwarding protocol for mobile wireless sensor networks will allow performance critical application requirements to be met in mobile deployments. Our use of machine learning will allow our diversity exploiting designs to work in any real deployment.
1.3 Dissertation Organization

The rest of this dissertation is organized as follows: Chapter 2 presents a detailed survey of related work on sensing diversity for performance critical applications in distributed sensor networks and body sensor networks. In Chapter 3, we present Watchdog, a machine-learning based approach to meeting user performance requirements by addressing sensing diversity. In Chapter 4, we expand on our learning-based approach to present Wolfpack, a distributed method to exploiting sensing diversity which also focuses on identifying how and when sensor collaboration is needed to meet user performance requirements. Chapter 5 presents a practical solution to activity recognition for body sensor networks, providing a user friendly solution which identifies and uses only the most helpful sensors and provides retraining detection without the need for ground truth. In Chapter 6, we present Remora, which exploits physical proximity among neighboring body sensor networks to provide increased accuracy and energy savings for daily activity recognition. In Chapter 7, we present Sidewinder, a predictive data forwarding protocol for mobile wireless sensor networks which allows end users to receive sensing decisions in mobile environments. Finally, we present conclusions and future work in Chapter 8.
Chapter 2

Related Work

In this chapter, we discuss state of the art related to learning and exploiting sensing diversity for performance critical applications. In Section 2.1, we first describe existing work for learning the sensing capability of a deployment and meeting user or application accuracy requirements (confidence). Then, in Section 2.2, we discuss state of the art for body sensor networks with respect to providing a user friendly, lightweight, and accurate solution as well as discuss exploiting neighbor proximity for accuracy and energy gains.

2.1 Exploring Sensing Diversity and Distributed Sensor Networks

In this section, we discuss existing distributed sensor network approaches for exploring and capturing sensing diversity as well as routing detection decisions in mobile environments. First, in Section 2.1.1, we discuss distributed sensor network approaches for sensing and classification which are unable to explore the detection capability of a specific deployment,
meet user accuracy requirements, or both. In Section 2.1.2, we discuss existing sensor network and mobile ad hoc network approaches to data routing in mobile environments, none of which can address the volatile topology changes present when using low power radios.

2.1.1 Diversity Awareness and Meeting User Requirements

Some works ignore both sensing confidence and diversity. These include $k$-coverage approaches [133] [124] [1] [53] [75] that rely on $k$ nodes to be awake within the sensing range of a target location. In [29] [88], multiple modalities collaborate to detect events along with a sleeping scheme to save energy. Similarly, disc-based sensing models [81] [19] [34] [123] [111] address neither sensing confidence nor diversity.

Other work attempts to provide sensing confidence and meet user accuracy requirements through theoretical modality-specific sensing models and data fusion-based [118] collaboration. However, due to their lack of complexity, these models do not address sensing diversity and make heterogeneous sensor collaboration difficult. A structural health monitoring system for accelerometers is presented in [45] and a camera-based detection coverage approach is presented in [58]. Specific sensing models for coverage with acoustic, seismic, and infrared sensors are presented in [43]. Signal attenuation-based models are described in [112] [121] [13] which give false positive and false negative rates for a given modality and training data set, allowing for data fusion between a cluster of sensors. Another collaboration scheme using a signal attenuation model [134] incorporates a sleeping scheme to save energy. A noise distribution model is used for event detection in [102] and [122].

Some approaches attempt to address sensing diversity by accounting for sensing dif-
ferences in different sensors but cannot provide sensing confidence. This includes works [56] [116] that use a similarity metric to ensure enough nodes are awake within the sensing range of a target location as well as those [117] that calibrate sensors based on differences in their readings. Some approaches use machine learning to provide collaboration [28] [33] [84] [113] [11] [61] [138], but these works do not fully explore the effects of sensing diversity on collaboration nor do they provide sensing confidence. Other works [65] address sensing diversity in providing sensing confidence but do not provide lightweight and decentralized collaboration appropriate for wireless sensor networks.

2.1.2 Data Forwarding in Mobile Environments

In general wireless ad hoc and sensor networks, a group of existing routing protocols perform a one-way [129] [92] or two-way [97] [38] [59] [72] [57] path discovery and use the discovered path for consecutive data communication. A sensor network has a much smaller radio range, typically 10 ~ 40m, compared to a general wireless ad hoc network, 150 ~ 250m. When excessive topology changes are observed in a sensor network, continuous maintenance of fixed routing paths becomes impractical for supporting effective communication. The approach in [10] provides a mobility-induced time and space adaptive beaconing mechanism to provide destination location information, but routing decisions are only made at each hop. We illustrate the detrimental effects of these topology changes on traditional wireless ad hoc protocols in Section 7.1, which motivates the need for a protocol that makes routing decisions based on information accumulated at each hop to ensure data reaches the sink.

Another group of routing protocols [63] [70] [108] [48] [37] use periodic beaconing messages to discover neighbors, and use neighbors' geographic locations for local forwarding to
achieve multi-hop communication. Some solutions exist to modify the neighbor table in the mobility case, such as [114] and [10]. When geographic locations are not available, some variants [93] [105] [17] use virtual coordinates or landmarks compared to selected anchor nodes for local forwarding. Like [10], [14] also provides a mobility-induced time and space adaptive beaconing mechanism to provide destination location information, but still makes use of a neighbor table. [49] [77] [144] use zone-based forwarding to address mobility, which still suffers in highly mobile environments. In Section 7.1, we show through quantitative study how highly mobile environments cause traditional wireless sensor routing protocols to fail. This illustrates the need for a dynamic routing protocol with low overhead that does not rely on a neighbor table or fixed routes.

2.2 Sensing Diversity and Body Sensor Networks

In extending our sensing diversity exploitation to body sensor network applications for activity recognition, we discuss two important bodies of related work. First, in Section 2.2.1, we demonstrate that existing approaches are lacking with respect to a user friendly solution, lightweight and accurate classification, and adaptation to dynamics present in body sensor networks. Then, in Section 2.2.2, we show that existing body sensor network and mobile applications do not exploit physical proximity to provide significant increases in both accuracy and energy savings.

2.2.1 Providing A Practical Solution

Many methods perform classification with multiple on-body sensor nodes but have no mobile, on-body aggregator for sensor control and activity recognition feedback. Some works
use multiple on-body sensor motes to detect user activities, body posture, or medical conditions, but such motes are only used to collect and store data with analysis performed offline. Such approaches limit user mobility due to periodic communication with a fixed base station and also lack real-time analysis and feedback to the user.

Other approaches use mobile phones for sensing and classification, but require the use of backend servers or offline analysis to train or update classification models. The authors of [89] provide model training with a short amount of initial data, but model updating is performed using a backend server. In [90], model training and classification is split between lightweight classifiers on a mobile phone and more powerful classifiers on a server. The authors of [96] present a sensor mote with an SD card attachment for interface with a mobile phone but do not provide an application which uses such a device. Mobile phone-like hardware is used in [33] to perform pothole detection, but data is analyzed offline.

Some works use a limited set of sensor modalities or use a separate classifier for each sensing modality, making classification difficult for deployments with a large number of heterogeneous sensors. Some works [5] [69] focus extensively on energy aware sensing models specifically for localization or location tracking. In [86], while the authors provide an adaptive classification approach, they only make use of a mobile phone microphone to recognize user context and activities, thus eliminating a wide range of user activities that are not sound dependent. The authors of [87] provide a component-based approach to mobile phone-based classification for different sensors and different applications, but each sensor component requires a separate classifier implementation. One approach uses both motes and mobile phones to perform fitness measurement [32], but simple sensing models are used that will not work for more general activity recognition applications. Activity recognition
and energy expenditure is calculated in [2] using an accelerometer-specific sensing model for on-body wireless nodes.

Lastly, several activity recognition methods do not provide online training to account for environmental dynamics or poor initial training data. The authors of [78] use AdaBoost to perform activity recognition but use custom hardware with very high sampling rates. In [94], the authors also use AdaBoost for activity recognition with mobile phones, but focus mainly on ground truth labeling inconsistencies and do not provide a practical system for long term use. The authors of [125] focus on duty cycling mobile phone sensors to save energy, but provide a rigid rule-based recognition model that must be defined before runtime. A speaker and emotion recognition system using mobile phones is presented in [104] which implements adaptive sampling rates but the classification models used are trained offline.

2.2.2 Inter-BSN Resource Sharing

Several collaborative sensing and classification approaches directly share resources among users, but none use sharing to achieve both high accuracy and energy efficiency. In [73], nearby drivers exchange traffic light data to determine optimal driving speed. Speaker recognition classifiers are combined among phones in physical proximity to each other in [89], which increases accuracy. However, an expensive classification method is used which requires the use of a backend server for training. Significant overhead is consumed transmitting trained classifiers from the server to the phone as well as when combining classifiers among phones.

Other collaborative approaches do not directly share sensing resources; all information is relayed or processed using backend servers. This approach is used in collaborative ap-
proaches for video editing [8], group activities [7], and a generic opportunistic framework [25]. To increase accuracy, one approach [76] shares classifiers among users with similar behaviors. Other works which use backend servers [103] [40] provide sensing quality tradeoffs, such as an adaptive sampling rate, to save energy.

Many existing on-body sensing and activity classification approaches do not allow any collaboration among users. On-body sensors are used for classification [2] [83] [20], some of which [67] [109] provide energy saving methods. Other approaches [86] [87] use only smartphone sensors for activity classification. A phone-only classification technique [23] provides an explicit energy-latency-accuracy tradeoff, while other smartphone methods [90] [125] [103] achieve energy savings with adaptive sampling.

Other works investigate interactions between multiple subjects but do not address the data or resources being shared as well as accuracy or energy concerns related to such resources. These include user proximity [31] [79], intercontact time [52] [62], mobility prediction [22] [94], and protocols for proximity-based mobile device pairing [95] [60].
Chapter 3

Watchdog

Many wireless sensor network applications, such as those for military surveillance [47], environmental and wildlife monitoring [91] [30], as well as vehicle tracking [28] require high accuracy and long system lifetimes. When a performance critical application makes detection or classification decisions that meet a user's accuracy requirement in terms of desired false positive and false negative rates, we say it is confident. Existing approaches to event detection and classification ignore the challenges of addressing sensing diversity and meeting user requirements. For example, sensing coverage approaches [133] [56] ignore the sensing capability differences among different sensors. Modality-specific sensing models [19] [136] make collaboration difficult in heterogeneous sensor deployments. Other approaches use machine learning [11] or aggregation [27] to capture differences in capability among different sensors but do not provide confidence.

To address these issues, we present Watchdog, a confident sensing framework for performance critical applications. We focus on confident event detection at a critical point, such as vehicle detection at a fixed location or intrusion detection. With Watchdog, we explore
the detection capability of different sensors and sensor clusters and choose the right sensor clusters to provide confident sensing. Our research results have been published in [65], and in this chapter, we answer the following research challenges:

- Exploring and exploiting the detection capability of a deployment. We explore the detection capability of a specific deployment and choose the right sensors to save energy and meet accuracy requirements.

- A generic solution. We provide a generic solution that works with a wide range of deployments, sensor modalities, and machine learning methods.

- Adaptation to environmental dynamics. We provide collaboration between higher power and lower power sensor clusters so that user requirements are met with only the necessary detection capability. We update sensor clusters and their machine learning detection models when no cluster can meet the user requirements.

The rest of this chapter is organized as follows: We first motivate our Watchdog design in Section 3.1 and describe our detailed Watchdog design in Section 3.2. We then present a performance evaluation in Section 3.3. Finally, we present conclusions in Section 3.4.

3.1 Motivation

In this section, we demonstrate the need for a new approach to confident event detection with reduced energy consumption by showing that performance differences among different sensors and sensor clusters cannot be ignored. Our goal is to provide confident event detection at a critical point, such as monitoring vehicular traffic flow, detecting soldiers
crossing a bridge, or detecting natural disasters, such as an earthquake. As an example, we use the Wisconsin SensIT experiment [28] to perform vehicle detection at a specific location. The SensIT experiment consists of a 23 node network with acoustic, seismic, and infrared sensors. Vehicles make 20 passes along a road through the network with ground truth provided by a GPS trace. Trace data of raw sensor energy is provided for each sensor at a raw sampling rate of 4960Hz. We provide this unmodified real sensor data and ground truth as input to a Java-based trace-driven wireless sensor network simulation run on a PC. While the sensor data and ground truth is real, we simulate communication for low power mote class devices with 802.15.4 radios, such as the Crossbow IRIS [24]. While we are aware that radio communication can be lossy in wireless sensor networks, in this chapter we focus on sensing accuracy, not communication quality, and assume reliable communication.

Using the trace data, we define a target location at the "X" along the road in Figure 3.1. Data is aggregated into time intervals of 100ms length for a total length of 6763 intervals. At each interval, we classify sensor and sensor cluster data into events when the vehicle is present within 2 meters of the target location and non-events when the vehicle is farther away or not present. With this in mind, we determine vehicle detection accuracy

![Figure 3.1: Event detection performance with vehicle trace data. The target location is marked by the "X" on the road.](image-url)
for individual sensors and sensor clusters using the method we present in Section 3.2.3 and we plot the results in Figure 3.1, highlighting the impact on existing solutions.

In Figure 3.1 (a), we first observe that sensors with the same distance to the target location may exhibit different detection accuracies. For example, nodes 41 and 50 are both 80m from the target location, but their detection accuracies are different, 93% and 56%, respectively. This is because while accuracy generally decreases with distance from the target location, terrain changes and environmental conditions still produce irregularities in sensor performance, which is consistent with the findings in [56]. This observed sensing irregularity can cause modality-specific sensing models to suffer, such as [136]. For example, a signal attenuation model for acoustic sensors [136] derives the same acoustic signal receiving power for sensors with the same distance to the target location. Therefore, the same detection accuracy is statistically derived for nodes with the same distance (node 41 and 50 in our example). This signal attenuation model cannot articulate the accuracy differences among sensors, such as determining which sensor is 93% accurate and which is 56% accurate in our example. For this reason, the system performance suffers and the required detection accuracy can not always be met, which we further demonstrate in Section 3.3.3.

In Figure 3.1 (a), we also observe that not all sensors within the 25m sensing range provide the same detection accuracy. For example, even though both node 60 and 54 are within the 25m sensing range of the target location, they have different detection accuracies, 93% and 86%, respectively. This observed sensing difference can cause sensing coverage-based schemes to suffer. For example, in [47], only one of multiple sensors with the sensing range is enabled at a time to provide sensor coverage (or 1-coverage) for energy savings. For our example, this means that either node 60 or 54 can be turned on to provide such
1-coverage. However, it is clear that using node 60 will provide 7% points better accuracy than with node 54. Unfortunately, sensing coverage schemes have no knowledge of such subtle but important detection accuracy differences, and hence cannot provide confident event detection.

Figure 3.1 (b) illustrates that different sensor clusters are able to provide the same detection accuracy. For example, clusters $C_1$, $C_2$, and $C_3$ (consisting of different sensor modalities) can all provide 100% detection accuracy even though individual sensors cannot. As shown in [131], clustering sensors can produce a synergistic effect, allowing sensors with complimentary detection strengths in different scenarios to collaborate. Exploring the detection capability of a deployment by evaluating the performance of different sensor clusters allows the most energy efficient clusters to be chosen to confidently detect events. However, such thorough exploration is not achieved by existing works and thus user-defined accuracy requirements cannot be met with reduced energy usage.

From the trace data analysis, it is very clear that existing approaches have difficulty meeting user required detection accuracy. This is due to lack of detailed detection accuracy knowledge of individual sensors and sensor clusters. Therefore, it is imperative to design a scheme that can provide confident event detection with user-defined accuracy, address in-situ sensing reality, and reduce energy usage.

### 3.2 Watchdog Design

In our Watchdog architecture, depicted in Figure 3.2, computationally limited nodes with sensors are connected through a wireless link to a more powerful aggregator, such as a
mobile phone. Nodes collect sensor data and return observations to the aggregator which makes event detection decisions. Our architecture is structured to solve the challenges that arise from providing confident event detection through the use of the modules we describe below and elaborate upon in the following subsections.

The Local Aggregation module, located on sensor nodes, is used to provide efficient collaboration between heterogeneous sensors. Sensor data is aggregated such that observations from different sensor modalities can be compared and easily fused at the aggregator to make cluster-level detection decisions.

In Cluster Generation, we explore the detection capability of a deployment by determining the detection capabilities of individual sensors and sensor clusters within the deployment. We use machine learning to perform event detection and determine training accuracy of heterogeneous sensor clusters. In Section 3.2.2, we apply several machine learning techniques to our generic design which we also evaluate in Section 3.3.

In Sentinel and Reinforcement Selection, clusters are selected that meet user detection requirements and adapt to changes in environmental dynamics. Using the deployment detection capability determined by Cluster Generation, a subset of that capability is selected such that the user requirements can be met. A cluster of low-power sentinel sensors is selected to meet the user detection requirements for many runtime observations, when
event detection decisions are easy. For more difficult event detection decisions where more
detection capability is needed, a cluster of reinforcement sensors is selected to ensure the
user detection requirements are met.

In Runtime Event Detection, the detection capability is adapted to runtime observations
using the clusters selected in Sentinel and Reinforcement Selection. Specifically, a low-
power set of sentinel sensors make easy event detection decisions to meet user accuracy
requirements. When the sentinel sensors determine that more detection capability is needed,
a second set of reinforcement sensors are used to make a confident detection decision.

With Online Retraining, described in Section 3.2.6, Watchdog is able to detect significant
changes in environmental dynamics which cannot be captured by the existing sentinels and
reinforcement clusters. When such changes are detected, new observations are labeled with
ground truth and new clusters are generated to ensure user requirements are met. We now
describe each design module in detail:

3.2.1 Local Aggregation

On a sensor node, the Local Aggregation module allows nodes to aggregate data locally
at regular intervals, allowing for reduced radio communication and heterogeneous sensor
fusion. The module is flexible to allow incorporation of different widely used aggregation
algorithms. In our vehicle detection scenario, for each sensor, our aggregation method
returns the normalized sample variance of the raw sensor data every 100ms. The aggregation
interval length is selected such that an event can be captured.

For sensor $j$, and aggregation interval $t$, aggregated sensor data is represented as obser-
vation $O_{j,t}$. The aggregator fuses observations from each sensor $j$ in a sensor cluster $C_i$ to
form an observation \( O_{C_{i},t} \) for that cluster. The fused observations can then be used by the aggregator to determine sensor cluster accuracy or make runtime detection decisions. We describe how sensor observations are fused together in Section 3.2.2, as different approaches are used for each detection algorithm.

**Transmission Energy Savings.** From training observations, a default observation value is determined for each sensor, which is associated with non-events. To save energy, a node only transmits observations when at least one of its sensors makes a non-default observation. At each aggregation interval, if the aggregator does not receive an observation from a sentinel or reinforcement sensor, it assumes the default observation value.

Since our original approach for transmission energy savings in [65] requires discrete sensor data, we expand on this to allow for machine learning methods that continuous data as input. We implement a bandwidth and energy saving approach via a transmission threshold defined for each sensor. Using training observations and labeled ground truth for each observation, each sensor computes a non-event and event centroid during initial training and when clusters are updated. The non-event centroid values are transmitted to the aggregator during the cluster generation process.

![Figure 3.3: Transmission threshold 0 ≤ α ≤ 1 for a cluster member sensor. Sensor readings below α are never transmitted and assumed by the aggregator to be the non-event centroid, while readings greater than or equal to α are always transmitted.](image)

A system-defined transmission threshold, \( \alpha \), depicted in Figure 3.3, resides between the
non-event and event centroids for each cluster member sensor. If a member sensor reading falls below the threshold, the reading is not transmitted to the aggregator. Therefore, if the aggregator does not receive a data sample from a cluster member sensor, it assumes the non-event centroid reading for that sensor when forming a cluster observation. We show the effectiveness of this threshold in our evaluation in terms of energy consumption and accuracy.

3.2.2 Machine Learning Exploration

To discriminate events from non-events, Watchdog can make use of many machine learning techniques, however, we focus on Hidden Markov Models, $k$-means clustering, and Fisher's Linear Discriminant, which we compare in our evaluation. We first explain Hidden Markov Models with uniform discretization. We then illustrate how to use $k$-means clustering to discretize a vector of real-valued sensor readings as input into a discrete Hidden Markov Model. Next, we explore the use of a continuous Hidden Markov Model. Lastly, we look at Fisher's Linear Discriminant as another approach for event detection and improving detection accuracy to meet user requirements.

**Hidden Markov Models with Uniform Discretization.** Hidden Markov Models require little initial configuration and are built upon the premise of determining hidden states (events) from a sequence of known observations (sensor readings) [101]. HMMs assume that events and non-events are correlated with time and make use of transition probabilities to further predict the likelihood of an event at each aggregation interval. HMMs also allow aggregated data from different sensor modalities to be easily fused, providing a generic framework that is adaptable to many application scenarios.
The HMMs we use have 2 hidden states: $E = 0$ for non-events and $E = 1$ for events. The HMM also has $m$ possible observations. To form HMM observations for a sensor cluster $C_i$, aggregated observations for each sensor in the cluster are independently discretized into one of $m$ evenly sized bins. For a training or runtime aggregation interval $t$, discretized observations for each sensor are then fused into a single discrete observation $O_{C_i,t}$ which is the average bin value of all observations in the cluster. Using a fused training observation sequence $O_{C_i}$, a cluster HMM for a cluster $C_i$ is trained using the Baum-Welch algorithm [101] to form state transition probabilities and observation probabilities. In our vehicle detection scenario, with 1000 training observations (100 sec), the Baum-Welch algorithm would converge within about 20-30 iterations for each cluster (the other HMM techniques we explore also exhibited similar convergence).

At each runtime aggregation interval $t$, a cluster HMM uses the Forward algorithm and a history of fused observations to determine an event probability $\gamma_t \in [0, 1]$. If $\gamma_t \geq 0.5$, the HMM determines that an event has occurred ($E_t = 1$), otherwise the HMM determines there is no event ($E_t = 0$).

**Clustering-based Discretization for HMMs.** In [68], clustering real-valued input data into discrete observations yields significant accuracy improvements over uniform discretization when using discrete Hidden Markov Models. In Watchdog with clustering-based discretization for HMMs, we use $k$-means clustering [12] to discretize data from a sensor cluster into one of $m$ observations required by the discrete HMM. For each sensor cluster formed during Cluster Generation, the aggregator creates $m$ centroids using $k$-means clustering of recently collected sensor cluster reading tuples. Unlike the approach used with uniform discretization, each observation input $O_{C_i,t}$ is a tuple of aggregated observations.
for each sensor in cluster $C_i$. For each tuple of sensor readings during Cluster Generation or Runtime Event Detection, the aggregator determines which of the $m$ centroids is closest to the tuple. For a cluster $C_i$, the centroid index is used as the current observation input to the Baum-Welch algorithm for HMM training or the Forward algorithm for Runtime Event Detection.

**Continuous HMMs.** In using a continuous Hidden Markov Model, we remove the requirement for discretization of sensor data. A cluster of sensors has a large number of possible reading combinations which must be discretized into a small number of observations. Using continuous Hidden Markov Models \[101\] with Gaussian distributions, we can preserve the granularity of the original sensor readings when providing training and runtime observation input. As with the clustering-based HMM, raw sensor readings for each generated cluster are collected by the aggregator and then fed as tuples into the Baum-Welch algorithm for HMM training or the Forward algorithm for detecting events in Runtime Event Detection. Also like the $k$-means approach, each observation input $O_{C_i,t}$ for a cluster $C_i$ at aggregation interval $t$ is a tuple rather than a single discrete value.

**Fisher’s Linear Discriminant.** While Hidden Markov Models can capture the correlation of events with time as well as the correlation of events with different cluster observations, HMMs cannot fully capture the data dependencies between different sensors in a cluster. To address this concern, we also implement Fisher’s Linear Discriminant \[12\], which attempts to find a projection (i.e. linear combination) of sensor cluster readings that maximizes the separation of event readings from non-event readings. As with clustering-based HMMs and continuous HMMs, the aggregator forms a tuple of aggregated observations from each sensor in a cluster to provide input for training and runtime detection. However,
like the uniform discretization HMMs, each individual sensor reading in an observation tuple \( O_{C_i,t} \) is uniformly discretized into one of \( m \) values. This discretization allows for the construction of discrete observation distributions for each sensor which are then used to partition event and non-event training data.

Once a partition is found during training that best separates the event observations from the non-event observations, all training observations are projected onto the line perpendicular to the partition to determine event and non-event centroids. New observations are then projected onto the same line and are classified by determining the nearest centroid. Like the other HMMs, we can also determine an event probability \( \gamma \) for an observation by comparing the distances between the projected observation and the centroids.

### 3.2.3 Cluster Generation

In Cluster Generation, we determine the detection capabilities of individual sensors and different sensor clusters, exploring the detection capability of a specific deployment. To do this, we generate sensor clusters of each possible size, ranging from size 1 for individual sensors to a cluster consisting of all available sensors. Using machine learning, we train a detection model and determine accuracy for each cluster using sensor training data labeled with event ground truth. We explain the cluster generation process in further detail using Algorithm 1.

With \( N \) sensors in a network, to completely explore the network detection capability, we ideally would generate all possible clusters from size 1 to \(|N|\). However, since computing resources are limited, we compute \( M \) random clusters of each possible size from which to choose sentinels and reinforcements. By computing a fixed number of clusters for each
Algorithm 1 Cluster Generation

Input: Set of all sensors in network $N$, user-defined false positive rate $ufp$ and negative rate $ufn$, training observations $O = \{O_{C_i,t}|C_i \subset N, 1 \leq t \leq T\}$, ground truth $G = \{G_t|1 \leq t \leq T\}$, number of clusters for each cluster size $M$

Output: Set of clusters $C = \{C_i|C_i \subset N\}$

1: Randomly generate $M$ clusters for each size $k(1 \leq k \leq |N| - 1)$, add to $C$
2: for all clusters $C_i \in C$ do
3:   Train event detection model for $C_i$ using $O_{C_i}$
4:   for Aggregation interval $t(1 \leq t \leq T)$ do
5:     Determine event probability $\gamma_t$ using $C_i$ and $O_{C_i}$
6:     if $\gamma_t \geq .5$ then $E_t = 1$ else $E_t = 0$
7:     Compare system event decision $E_t$ with $G_t$
8:     Update $fn(C_i)$, $fp(C_i)$, $fn(C_i, \gamma_t)$, $fn(C_i, \gamma_t)$
9:   end for
10: end for

size, our exploration approach is comparable in computational efficiency and effectiveness as more advanced feature selection approaches such as simulated annealing [71] [18].

For each generated cluster $C_i$ in the set $C$ of all generated clusters, we train a machine learning detection model. A model for each cluster is trained using a sequence of training observations $O_{C_i}$ and ground truth labels for each training observation $G$. Training observations for each sensor as well as ground truth are collected before runtime or for a short period during a runtime update.

With a trained detection model for each cluster, we can determine a cluster's event decision $E_t$ for each aggregation interval $t$. $E_t$ is derived from the cluster's event probability $\gamma_t$ at each training aggregation interval $t$. As previously explained, the cluster determines an event occurred at interval $t$ ($E_t = 1$) if $\gamma_t \geq .5$ and no event occurred ($E_t = 0$) if $\gamma_t < .5$. We can then use the cluster's event decision sequence $E = \{E_t|1 \leq t \leq T\}$ to compare with known ground truth $G = \{G_t|1 \leq t \leq T\}$ at each aggregation interval to determine cluster training accuracy. If, at aggregation interval $t$, the event detection decision is equal to the ground truth ($E_t = G_t$), then the cluster made a correct decision at $t$. Otherwise,
the decision was a false positive or false negative.

![Figure 3.4](image-url)

**Figure 3.4:** Event probability breakdown for a cluster $C_i$ with a 6% overall false positive rate and no overall false negative rate. For each .1 event probability range, the associated false positive rate $f_p(C_i, \gamma)$ and false negative rate $f_n(C_i, \gamma)$ are shown as bars. All ranges that have no observations yield a false positive or false negative rate of 1, since no accuracy can be determined for that range and hence we assume the worst.

**Event Probability Discussion.** We can compute the overall accuracy for each cluster $C_i$ by comparing all event detection decisions $E_t$ to ground truth $G_t$ to determine the overall false negative rate $f_n(C_i)$ and the overall false positive rate $f_p(C_i)$. However, a cluster with an overall low false positive or false negative rate may have all its incorrect decisions result from event probabilities that hover near .5. During runtime detection, it is likely that an event probability near .5 will result in an incorrect decision. Consequently, it is beneficial to differentiate the accuracies between event probabilities. During runtime detection, possible bad decisions made by sentinels due to middle-range event probabilities can be caught and reinforcements can be used to meet the user requirements.

To study the correlation between event probability and detection accuracy, for each cluster $C_i$, we break down each training event probability $\gamma_t$ into $p$ ranges of size $1/p$. For each range we compute false positive rates $f_p(C_i, \gamma)$ and false negative rates $f_n(C_i, \gamma)$. Figure 3.4 shows an event probability breakdown of a cluster $C_i$ from the Wisconsin vehicle trace data with 97% overall accuracy with $p = 10$ probability ranges. From the figure, it
is clear that all negative event decisions have an event probability in the $[0, .1)$ and $[.2, .3)$, ranges, while all event decisions have a probability in the $[.9, 1]$ range. During runtime detection, the event probability breakdown for the sentinel cluster is used to determine if an event probability $\gamma_t$ does not meet user false positive and false negative requirements and that reinforcement observations should be collected to make a confident decision.

### 3.2.4 Sentinel and Reinforcement Selection

With the deployment detection capability explored by determining accuracy for all generated clusters, we choose a subset of the deployment to remain awake during runtime detection as sentinels and reinforcements to make confident detection decisions. We choose sentinels such that all negative event decisions can be made with confidence: that the user’s false negative requirement is met by sentinels. Since communication is the most energy intensive operation in wireless sensor networks [110], we minimize energy usage by selecting a sentinel cluster with sensors on the fewest number of nodes, for only one radio transmission is needed to report observations from multiple sensors on the same node in one aggregation interval.

Since sentinels are only concerned with determining the lack of an event with confidence, we leave more difficult observations to the more powerful reinforcements when negative event decisions cannot be confidently made by sentinels. Therefore we choose reinforcements so that both the user’s false positive and false negative requirements are met. We also ensure that the combined sentinel and reinforcement clusters are located on the fewest number of nodes to save energy. The reinforcement cluster has at least one sensor that is not in the sentinel cluster in order to ensure there is some added benefit from sampling reinforcement
data. The sentinel and reinforcement selection algorithm is given in Algorithm 2.

**Algorithm 2 Sentinel and Reinforcement Selection**

**Input** Set of all sensors in network $N$, set of trained clusters $C$, user-defined false positive rate $ufp$ and negative rate $ufn$

**Output** Sentinel sensors $s$, Reinforcement sensors $r$

1: /*Sentinel Selection*/
2: $fn(s) = 1$; $s$.numNodes=$|N|$; $s = N$;
3: for all clusters $C_i \in C$ do
4: /*Meet user FN with least energy*/
5: if $fn(C_i) \geq ufn$ and $C_i$.numNodes$\leq s$.numNodes then
6: $s = C_i$
7: $s$.numNodes = $C_i$.numNodes
8: end if
9: end for
10: /*Reinforcement Selection*/
11: $fp(r) = 1$; $fn(r) = 1$; $r$.numNodes=$|N|$; $r = N$;
12: for all clusters $C_i \in (C - s)$ do
13: /*Meet user FP and FN with least energy*/
14: if $(s \cup C_i)$.numNodes$\leq r$.numNodes and $fp(C_i) \leq ufp$ and $fn(C_i) \leq ufn$ then
15: $r = C_i$
16: $r$.numNodes = $C_i$.numNodes
17: end if
18: end for

### 3.2.5 Runtime Event Detection

In Runtime Event Detection, sentinels and reinforcement sensors sample observations at each aggregation interval while all other nodes are asleep. The aggregator dynamically determines an event detection decision $E_t$ for each interval $t$ using sentinel or reinforcement observations, assuming a default observation value if no transmission is received. The Runtime Event Detection algorithm is described in Algorithm 3.

As shown in the algorithm, for each runtime aggregation interval $t$, sentinels determine an event probability $\gamma_t$ using the same method performed in Cluster Generation except runtime observations are used. If $\gamma_t < .5$, the sentinels can confidently determine that no event has taken place ($E_t = 0$) since the sentinels were selected such that the user’s false
**Algorithm 3 Runtime Event Detection**

**Input** Sentinels $s$, reinforcements $r$, runtime observation for $s$ for the current aggregation interval $O_{s,t}$, may also receive runtime observations for $r$ for the previous and current aggregation intervals $O_{r,t-1}, O_{r,t}$

**Output** Event detection decision for the current aggregation interval $E_t$ and for the previous interval $E_{t-1}$ if $E_{t-1}$=UNDECIDED

1: if $E_{t-1}$=UNDECIDED then
2: /*Make a confident decision at $t-1$ using $r$*/
3: Determine $\gamma_{t-1}$ using detection model from $r$ and $O_{r,t-1}$
4: if $\gamma_{t-1} \geq .5$ then $E_{t-1} = 1$ else $E_{t-1} = 0$
5: end if
6: Determine $\gamma_t$ using detection model from $s$ and $O_{s,t}$
7: if $\gamma_t < .5$ then
8: $E_t = 0$ /*$s$ confidently determines no event at $t$*/
9: else if $\gamma_t \geq .5$ and $fp(s, \gamma) \leq fp(u)$ then
10: $E_t = 1$ /*$s$ confidently determines an event at $t$*/
11: else if $\gamma_t \geq .5$ and requested $O_{r,t}$ has been received then
12: /*Make a confident decision at $t$ using $r$*/
13: Determine $\gamma_t$ using detection model from $r$ and $O_{r,t}$
14: if $\gamma_t \geq .5$ then $E_t = 1$ else $E_t = 0$
15: else
16: /*A confident decision cannot be made at $t$ using $s$*/
17: $E_t$=UNDECIDED; request $O_{r,t}$ and $O_{r,t+1}$
18: end if

negative requirement is always met. However, if $\gamma_t \geq .5$, we must check if the sentinels meet the user's false positive requirement for the given probability range in which $\gamma_t$ falls into, $fp(s, \gamma)$. If the user false positive requirement is met, $fp(s, \gamma) \leq uf_p$, the sentinels can confidently determine that an event has occurred ($E_t = 1$). Otherwise, when $fp(s, \gamma) > uf_p$, then the user false positive requirement is not met, $E_t$ is undecided, and more detection capability is required by requesting reinforcement observations. The aggregator sends a request message to retrieve reinforcement observations for intervals $t$ and $t + 1$ when a confident decision cannot be made by the sentinels. The reinforcement observations for $t$ will be returned at the end of interval $t + 1$. Piggybacking reinforcement observations for interval $t+1$ along with the observations for $t$ will allow the aggregator to use reinforcement observations to make a decision for $t+1$ if the sentinels are not confident for $t+1$. Another
reinforcement observation request message for interval $t + 1$ would not be necessary.

When sentinel observations are returned during an interval $t$ for the previous interval $t-1$, the aggregator can make a confident decision, since the sentinels meet the user accuracy requirements. $\gamma_t$ is determined using the reinforcement observations and an event, $E_{t-1} = 1$, is confidently determined if $\gamma_t \geq .5$. Otherwise, $E_{t-1} = 0$.

Figure 3.5: Runtime detection timeline with sentinel and reinforcement event decisions, where $ufn = ufp = 0.5$. Gray areas indicate sensor readings that trigger non-default observations. Aggregator-determined event probabilities are indicated by $\gamma_t$ and event decisions are indicated by $E_t$. Radio transmissions due to non-default observations are indicated by the arrows.

To illustrate Runtime Event Detection, an example is presented in Figure 3.5. In the figure, the sensors on node 1 are sentinels while the other two sensors on nodes 60 and 61 are reinforcements. During the first interval $t = 1$, no sensors report non-default observations, so the base station determines an event probability of .02. Since the sentinels have been determined to meet the overall false negative requirement, $fn(s) \leq ufn = .05$, the decision is confident. A similar decision also occurs at $t = 3$. At $t = 2$, the sentinels capture an event and report their observations via radio, yielding an event probability of .98. The false positive rate for sentinels when $\gamma_t = .98$ was determined during training as .02, so this is a confident decision ($.02 \leq ufp = .05$). At $t = 4$ the seismic sentinel sensor does not
capture the event, and the sentinel false positive rate for the current observation and event probability was determined from training as .45. Since .45 is greater than \( u_{fp} = .05 \), the aggregator could not make a confident decision and more detection capability is needed. Therefore, reinforcements are signaled to return their data for \( t = 4 \) at the end of interval \( t = 5 \). At \( t = 5 \), the reinforcement data yields a confident event decision for \( t = 4 \) since sentinels always meet the user requirements and the sentinel data determines that no event has occurred.

### 3.2.6 Online Retraining

During runtime, a sentinel or reinforcement cluster may experience a drop in detection performance, running the risk of not meeting the user detection requirements. Such a performance drop may be due to changes in background noise or to the properties of the event. In these cases, the sentinel and reinforcement clusters are disbanded and new, more accurate, sentinel and reinforcement clusters are regenerated. With Online Retraining, the aggregator receives periodic feedback as to the accuracy of detection decisions. This feedback can be provided in a manner similar to [67], where K-L divergence is used to compare training data to current runtime data at each aggregation interval, determining that an update is needed when runtime data is significantly different than training data for each active sensor.

When an update is triggered, the aggregator broadcasts an update message to notify all sensor nodes. Each awake node maintains a cache of the most recent readings for each sensor and upon receipt of an update message, these readings are transmitted back to the aggregator. The aggregator then collects new ground truth for the recent observations and
selects new sentinel and reinforcement sensors through the Cluster Generation and Sentinel and Reinforcement Selection processes described in Section 3.2.3 and Section 3.2.4, respectively. Runtime Event Detection then proceeds with the new sentinels and reinforcements as in Section 3.2.5.

A new, updated cluster may change with respect to the old cluster in three ways. First, the new cluster may consist of the same exact sensors as the old cluster only with a new detection model at the aggregator. Second, a newly formed cluster may also reside on the same nodes as the old but contain different sensors. Third, a new cluster may also reside on different nodes than the previous cluster. In future work, we will predict how a cluster changes during an update in order to reduce overhead in generating new clusters as well as to ensure energy fairness through load balancing among clusters.

After sentinel and reinforcement sensors are selected during initial training or an update, a subset of nodes is selected as candidate nodes from among all non-sentinel and non-reinforcement sensor nodes. Such candidate nodes remain awake and sample data so that during an update candidates may be selected to become sentinel or reinforcement sensor nodes if the current sentinel or reinforcement nodes cannot meet the user requirements.

3.3 Evaluation

Watchdog is designed as a generic framework, so we evaluate its performance in two different application scenarios using a PC-based Java simulation: vehicle detection using trace data and a building traffic monitoring application using IRIS motes. For the vehicle detection trace, we use the same simulation methodology described in Section 3.1. We use one pass of
the vehicle (70s) as training data and the remaining trace with 10 more passes as runtime data. In the building traffic monitor experiment, we place five IRIS motes with attached MTS310 sensorboards (2-axis accelerometer, 2-axis magnetometer, acoustic, light sensors) [24] on the main entrance door of an academic building to monitor the traffic pattern of when people are most often entering and leaving the building. We define an event and measure the ground truth as the time period during which someone opens the door and walks through (either entering or exiting), with the door automatically closing behind. We obtained ground truth via video recording of the building entrance and sampled data at 20ms intervals using the heterogeneous sensors on the mote sensorboards. We also use a 4s aggregation interval and 2 minutes of data for training.

We compare against a sensing coverage-based framework and a modality-specific sensing model using data fusion. The sensing coverage approach, V-SAM [56], is a state of the art scheme which in contrast to conventional coverage approaches, attempts to keep awake sensors that sample similar data. We also force k-coverage on V-SAM, where 1 to 3 nodes are awake to cover an event; only 1 node must detect an event for V-SAM to detect an event. We also compare Watchdog with a classical model-driven event detection solution [136] that uses a modality-specific sensing model. In [136], a signal attenuation model is used to estimate signal energy for targets of different distances with a Gaussian noise distribution model. Given user-defined false positive rate, the model-driven approach can derive an event detection threshold for the average energy readings of all sensors in a cluster.

In Section 3.3.1, we first demonstrate that Watchdog is able to explore the detection capability of a specific deployment and cluster the right sensors to meet user detection requirements. Next, in Section 3.3.2, we compare against a sensing coverage-based frame-
work and illustrate that Watchdog achieves a significantly higher performance. In Section 3.3.3, we compare against a data fusion-based modality-specific sensing model and show that Watchdog can adapt the detection capability to runtime observations and meet user detection requirements while the model-driven approach cannot. We explore the effects of different machine learning approaches in Section 3.3.4, and investigate in Section 3.3.5 how Watchdog can create new clusters when the existing sentinel and reinforcement clusters are unable to handle a significant environmental change. Lastly, we demonstrate the benefits of our transmission energy saving approach in Section 3.3.6. In the experiments, for Watchdog, we generate $M = 15$ clusters for each possible size. By default, we use HMMs with uniform discretization and Online Retraining disabled. We also set user requirements to 5% for false positives and false negatives.

3.3.1 Exploring Detection Capability and Meeting Requirements

Using the building traffic monitor trace, we show that by exploring the detection capability of a specific deployment, Watchdog can choose the right sensor clusters to meet user-defined false positive and false negative rates. Using the trace, in Figure 3.6 (a) (b), we plot the number of clusters for each cluster size that achieve the same training false positive or false negative rate. In Figure 3.6 (c) (d), we plot cluster training performance compared with runtime performance.

In Figure 3.6 (a) (b), there are only a limited and discrete number of false positive and false negative rates that the deployed system can support. To that end, a user can only require a false positive or false negative rate that can be supported by the system. For example, most sensors and sensor clusters have false positive and false negative rates near
Figure 3.6: Cluster Training and Runtime Detection. An integer besides the "x" denotes the number of clusters that give the corresponding FP or FN rate.

zero, while only a few experience false positive rates greater than 70% or false negative rates greater than 45%. This set of cluster performances is determined by the sensor hardware and local sensing reality where the system is deployed. Different scenarios may produce different false positive and false negative rates for each cluster.

In Figure 3.6 (a) (b), we also observe that even in a small deployment with "5 IRIS x 6 sensors each = 30 sensors", there are a large number of sensor clusters available to meet user specified false positive or false negative rate. As shown in Figure 3.6 (a), there are exactly 3+3+2=8 sensor clusters that demonstrate a 5% false positive rate in the training data and there are 189 sensor clusters in Figure 3.6 (a) that demonstrate smaller than a 5% false positive rate. So, in total, 8+189=197 different sensor clusters can be chosen to meet
the user-specified 5% false positive rate.

In Figure 3.6 (c) (d), we observe that during runtime detection, Watchdog is able to meet the false positive or false negative rate explored during training. For example, Figure 3.6 (c) shows that 48 clusters with a training false positive rate of 0% achieve this performance during runtime; Figure 3.6 (d) shows that 182 clusters with a false negative rate of 0% also demonstrate no false negatives during runtime. In Figure 3.6 (c) (d), we also observe that clusters with higher training false positive or false negative rates achieve significantly better runtime performance: 6 clusters with a training false positive rate of 72% achieve a runtime false positive rate of 10%, and 13 clusters with a training false negative rate greater than 20% achieve a runtime false negative rate of 5% or less.

To summarize, these data illustrate that Watchdog is able to cluster the right sensors to meet user requirements during runtime. Plus, many clusters of different sizes exist to meet user-required accuracy. This allows for freedom in sentinel and reinforcement selection to adapt the detection capability to environmental dynamics and maximize energy savings.

3.3.2 Comparison with V-SAM

Using the building traffic monitor trace, we compare Watchdog to a state of the art sensing coverage framework that addresses sensing irregularity, V-SAM [56]. Though V-SAM cannot provide guaranteed accuracy, we set the Watchdog user requirements to the lowest false positive and false negative rates determined from training. Evaluation results are presented in Figure 3.7 with 95% confidence intervals over 20 runs.

In Figure 3.7 (a) (b), We observe that Watchdog outperforms V-SAM in every configuration: all modalities, individual modalities, and varying levels of V-SAM coverage.
Although using higher $k$-coverage and similarity-based coverage helps improve V-SAM performance, it is always outperformed by Watchdog, which consistently demonstrates close to 100% detection accuracy in Figure 3.7 (a) and close to zero false negatives in Figure 3.7 (b). None of the Watchdog or V-SAM configurations experience statistically noticeable false positives, so false positive rates are not illustrated. Watchdog can consistently outperform V-SAM because Watchdog fully explores the detection capability of individual sensors and sensor clusters in a deployed system and cluster the right sensors to meet user requirements. However, V-SAM has no detailed knowledge of detection accuracy, so the most accurate sensors may be excluded while poor performing sensors may become involved in detection decisions.

Figure 3.7: Watchdog and V-SAM comparison for different modalities, levels of V-SAM coverage, and training lengths.
In Figure 3.7 (c), we observe that Watchdog is much more energy efficient than V-SAM. We compute energy to transmit or receive each byte of a 802.15.4 packet on CC2420 radios [21] using a TDMA-based scheme. We use the default payload of 28 bytes for each payload which is more than sufficient to carry aggregated data for all sensors on the transmitting node. As shown in Figure 3.7 (c), Watchdog energy consumption is relatively constant for all modalities and for each modality, hovering around $9 \times 10^{-4}$ J, since typically only 1-2 nodes are used in forming both sentinel and reinforcement clusters. However, V-SAM energy consumption (when achieving good performance) is much more varied: $10 \times 10^{-4} \sim 26 \times 10^{-4}$ J. While Watchdog may use more energy than 1 or 2-coverage V-SAM, Watchdog achieves about 35% points better accuracy compared with those V-SAM configurations. Watchdog is significantly more energy efficient than V-SAM since Watchdog fully explores the detection capability of individual sensors and sensor clusters. Hence, Watchdog can use this knowledge to adapt sensing capability to runtime observations while making confident detection decisions, but V-SAM cannot.

**Training Length.** In Figure 3.7 (d), we observe that for Watchdog to achieve the aforementioned superior detection accuracy and energy efficiency compared with V-SAM, only a short training length is needed. As shown in Figure 3.7 (d), when the training length increases, Watchdog performance improves quickly, surpasses V-SAM performance, and converges to near perfect accuracy after about 2 minutes, which is reasonably short for real applications. Even though V-SAM requires little training, which is invisible in Figure 3.7 (d), it demonstrates much lower detection accuracy and much higher energy usage than Watchdog. Since the training length is short, the use of periodic retraining can handle environmental changes.
3.3.3 Comparison with a Modality-Specific Sensing Model

In this section, we compare Watchdog with a classical model-driven event detection solution [136] that uses a data fusion-based modality-specific sensing model. For fair comparison, we use the same Wisconsin SensIT experiment trace data [28] used in [136] and make use of acoustic sensors to detect vehicles passing a static target location. Our evaluation is conducted in two scenarios: when the target location is well within the sensing range of all sensors, and when the sensors are located at the fringe of the detection range. In the first scenario, we use 5 acoustic sensors < 25m to the target location; in the second, we use 7 acoustic sensors with distances > 40m from the target location. The results are plotted in Figure 3.8.

![Comparison Diagrams](image)

**Figure 3.8:** Watchdog and modality-specific sensing model comparison with sensors located within 25m of, or more than 40m from, the target location.
For the <25m scenario, we observe from Figure 3.8 (b) that Watchdog always meets the user false positive requirement while the model-driven scheme cannot. For instance, in Figure 3.8 (b), the model-driven scheme has a 28% false positive rate when 20% is required, and gives a 42% false positive rate when 40% is required. We also observe from Figure 3.8 (a) that Watchdog yields perfect accuracy, while model-driven accuracy drops when the desired false positive rate increases. Watchdog performs better than the model-driven scheme because Watchdog always chooses sentinels and reinforcements that meet user requirements for confident event detection. The model-driven scheme does not exploit such subtle but important information.

For the >40m scenario, we also observe that Watchdog always meets user requirements but the model-driven scheme performs poorly or even fails. For example, when user requires a 5% false positive rate, the model-driven approach experiences very low accuracy, 67% in Figure 3.8 (a), and a very high false negative rate, 100% in Figure 3.8 (c). This is because for a low desired false positive rate, the model-driven detection threshold is set too high to detect any events. We also find in Figure 3.8 (c) that requesting higher false positive rates does not help much. The poor performance of the model-driven scheme and the good performance of Watchdog can be explained with the same reasons attributed to the <25m scenario.

Using the transmission energy model from Section 3.3.2, in both scenarios, Watchdog is found to consume significantly less energy than the model-driven scheme as shown in Figure 3.8 (d). This is because the model-driven scheme in [136] has a very simple energy saving scheme: nodes within the 25m “fusion range” are awake and nodes beyond the range all sleep. On the contrary, Watchdog adapts the detection capability to runtime
observations through the use of sentinels and reinforcements for more aggressive energy savings. In Figure 3.8 (d), we also observe that the model-driven scheme consumes more energy in the >40m scenario than the <25m scenario. This is because 7 nodes are used instead of 5.

Table 3.1: Adapting detection capability with reinforcements.

<table>
<thead>
<tr>
<th>Sentinel FP/FN (%)</th>
<th>Reinforc. FP/FN (%)</th>
<th>Reinforc. Requests (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.5/0.0</td>
<td>0.0/0.0</td>
<td>21</td>
</tr>
</tbody>
</table>

Adapting Detection Capability. Using the <25m scenario we illustrate in Table 3.1 how Watchdog adapts the detection capability to environmental dynamics. With desired false positive and false negative rates of 0%, a sentinel cluster is selected with a 9.5% false positive rate and 0% false negative rate. A more powerful reinforcement cluster is selected with a 0% false positive and false negative rate. During runtime, 79% observations are comparatively easy and hence confident decisions are entirely made by sentinels. When the sentinels make a decision that does not meet user requirements (for the 21% more difficult observations), reinforcements are used to make a confident decision. The reduction in radio transmissions made by using only the sensors necessary to meet user requirements ensures significant energy savings.

3.3.4 Machine Learning Comparison

Using the vehicle detection trace, we choose 79 target locations along the road and cluster sensors within a 100m range. More target locations increases the environmental dynamics and allows us to better investigate the effects of different machine learning algorithms as well as Online Retraining in Section 3.3.5. Figure 3.9 a) shows that Watchdog without
Online Retraining using Fisher’s Linear Discriminant can meet the user requirements for 95% of the target locations, while the modality-specific sensing model and V-SAM meet the user requirements for 63% and 85% of the target locations, respectively. A similar trend can be seen when comparing average accuracy for all target locations in Figure 3.9 b). With respect to the different machine learning methods, Watchdog with Fisher’s Linear Discriminant exhibits the best performance since it can determine the data dependencies between different sensors in a cluster and use the strengths of each sensor to maximize detection performance. Watchdog with a clustering-based discretization HMM is also able to improve on the default uniform discretization HMM since k-means clustering aids the HMM in separating events from non-events. The continuous HMM meets many fewer target
locations compared with other Watchdog implementations since it requires a much longer training dataset to train accurately.

Figure 3.9 b) shows that a slight accuracy increase allows Watchdog to meet user accuracy requirements for many more locations, making Fisher's Linear Discriminant especially valuable. For example, the continuous HMM without Online Retraining has over 90% average accuracy for all locations, yet only meets user requirements for 34% of these locations. However, with 98% average accuracy, Watchdog with Fisher's Linear Discriminant and no Online Retraining meets user requirements for 95% of locations. Similarly, increasing coverage for V-SAM increases its detection accuracy and the number of locations that meet the user requirements, but even 4-coverage V-SAM meets the user requirements for only 85% of locations. Nearly all of the detection errors for Watchdog, the modality-specific sensing model, and V-SAM are false negatives as shown in Figure 3.9 c); the average false positive rates for all approaches are less than 1% except the continuous HMM, which has an average false positive rate of 14%.

![Graph showing energy usage](image)

**Figure 3.10:** Energy usage.

In Figure 3.10, we show the average energy consumption for each approach across all target locations. We extend energy usage to include not only radio energy but also energy consumed by motes while awake and sensing, with details given in [110]. Watchdog con-
sistently experiences the lowest energy consumption, ranging from just over 0.2J with the uniform discretization HMM without Online Retraining to 0.4J for the continuous HMM with Online Retraining. Watchdog is able to reduce energy usage compared with other approaches since it chooses only the sensors and nodes needed to meet user requirements. V-SAM has much higher energy consumption, ranging from 0.5J for similarity coverage to 0.6J for 4-coverage. As coverage levels increase, more nodes are awake, increasing V-SAM energy usage. Furthermore, the standard deviation is higher than Watchdog, for when events are detected, V-SAM awakens all nodes to monitor the event until it is no longer detected. The modality-specific sensing model keeps all nodes awake at all times and has very high energy usage, 1.4J. Its standard deviation is also much higher since some locations have many nodes within the 100m fusion range, while other locations have few nodes.

### 3.3.5 Online Retraining

During runtime, Watchdog with Online retraining is able to handle significant environmental changes by retraining sentinel and reinforcement clusters. The 79 different detection locations we choose in the vehicle detection trace provide significant environmental dynamics, as the vehicle path varies widely with each pass. Watchdog configurations with Online Retraining have 2 candidate nodes awake at each time interval and the machine learning approaches that use a transmission threshold have the threshold set to $\alpha = 0.5$. In Figure 3.9 a), Online Retraining is able to increase the number of locations which meet user requirements for each machine learning method, ranging from 3% points for Fisher's Linear Discriminant to over 30% points for the continuous HMM. Similar increases in accuracy are also observed in Figure 3.9 b). Figure 3.10 shows that Online Retraining requires slightly
more energy due to communication overhead in forming new clusters and awake candidate nodes, but performance is still significantly better than the modality-specific sensing model and V-SAM.

Figure 3.11: Reinforcement usage and Online Retraining.

In Figure 3.11 a), we show that more powerful reinforcements are requested sparingly, regardless of the machine learning approach or the use of Online Retraining. Using Fisher's Linear Discriminant, some locations request reinforcement data more than 10% of the time. Fisher’s Linear Discriminant requests reinforcements slightly more than the other configurations for it can closely capture the data dependencies among sensors in a cluster and better determine when the sentinel cluster cannot make a confident detection decision. This adaptation in detection between clusters of differing sensing capability demonstrates the ability
of Watchdog to reduce energy consumption in comparison with other approaches.

Using Online Retraining, Figure 3.11 b) illustrates the total number of updates for each target location and each Watchdog machine learning configuration. The figure shows that a small number of updates allows Watchdog to maintain its high detection accuracy and meet the user requirements. Most target locations require no updates with all but the continuous Hidden Markov Model, for which most locations require 2, 3, or 4 updates. For Fisher’s Linear Discriminant and the discrete HMMs, the initial training is good enough to meet the user requirements for most locations, however since the continuous HMM does not receive enough training data, it requires more updates. Figure 3.11 b) also depicts average accuracy for target locations with each number of updates. As the number of updates per location increases, average accuracy generally decreases, reaching as low as 70% for the continuous HMM. Although locations with the largest amount of environmental change experience lower accuracy and more updates, without such updates accuracy would be even worse for these locations, as illustrated in Figure 3.9 b).

Figure 3.11 c) shows how sentinel and reinforcement clusters are reformed during an update. In the figure, the Watchdog configurations with the fewest total updates also experience the highest average accuracy among locations that have updates. Fisher’s Linear Discriminant has 51 updates in total and experiences an average accuracy of 98% for locations with updates while the continuous HMM has nearly 250 updates with an average accuracy of 94%. With Fisher’s Linear Discriminant, accuracy for most locations is high enough to meet the user requirements and few, if any, updates are needed. Also from the figure, over half of all updates for each configuration require formation of clusters with new nodes that were not members of the previous clusters. Environmental changes cause the
current nodes and sensors to fail to meet the user requirements, which prompts the creation of new clusters with new nodes that can address these changes. Thus, candidate nodes are also heavily used during updates.

3.3.6 Transmission Energy Savings

![Figure 3.12: Energy use and accuracy for different transmission threshold values.](image)

In Figure 3.12, we justify the choice of the transmission threshold $\alpha = 0.5$ in terms of a tradeoff between accuracy and energy usage for the machine learning approaches that use continuous valued input. We also disable the transmission threshold and show the impact of accuracy and energy when all readings are transmitted. Using the vehicle detection trace data, Figure 3.12 a) shows that total energy is affected little by increasing $\alpha$ and transmitting fewer readings. Accuracy remains relatively stable as $\alpha$ increases until $\alpha > 0.5$, when accuracy drops by as much as 12% points for Fisher’s Linear Discriminant and the continuous Hidden Markov Model. Figure 3.12 a) also shows that for configurations with Online Retraining, candidate nodes consume a significant amount of the total energy budget, for two additional nodes are awake during Runtime Event Detection.

While the transmission threshold may not have a significant impact on total energy, Figure 3.12 b) shows that the transmission threshold noticeably affects transmission en-
ergy. When all sensor readings are transmitted, all Watchdog configurations consume about 0.065J of transmission energy with while the highest transmission threshold setting all configurations consume about 0.2J. Energy decreases noticeably for only small and large $\alpha$, indicating that most sensor readings are closest to the event and non-event centroids and that most detection decisions can be easily made by the sentinel cluster, as is illustrated in Figure 3.11 a).

3.4 Conclusion

Existing works do not provide a holistic solution with respect to clustering the right sensors for confident event detection, heterogeneous deployments, and adaptation to environmental dynamics. Consequently, we present Watchdog, a generic event detection framework which can function in a wide array of applications and deployments. Unlike existing approaches, Watchdog can obtain the detection capability of a specific deployment and use this knowledge to cluster the right sensors to perform confident event detection. With a short training length, Watchdog chooses sentinel and reinforcement sensors which adapt the detection capability to confidently detect events while saving energy. We propose several different machine learning techniques which Watchdog can use to perform confident event detection and allow these methods to adapt to environmental changes over time. Our evaluation demonstrates that Watchdog largely exceeds the detection accuracy of existing approaches with reduced energy consumption. Our evaluation also demonstrates that Watchdog always meets user detection requirements when in many cases existing approaches cannot.
Chapter 4

Wolfpack

In Chapter 3, we introduce Watchdog, a confident sensing framework for event detection at a critical point, such as vehicle detection at a fixed location or natural disaster detection. We now expand upon this approach, exploring and exploiting sensing diversity to provide confident sensing in a distributed manner appropriate for low power wireless sensor networks. Many existing works ignore sensing diversity, the sensing capability differences among different sensors in a deployment, and assume all sensors have similar performance [132] [133]. Other works attempt to overcome sensing diversity through sensor calibration [117] or machine learning [28] [33] [83] but do not meet user accuracy requirements.

In this chapter, we take advantage of the sensing performance differences present in a deployment to meet user accuracy requirements. Through the use of real trace data for vehicle detection, we first capture and explore sensing diversity and identify the impact of sensing diversity on collaboration for confident sensing. We also explore two machine learning techniques appropriate for low power wireless sensor networks. Through theoretical analysis and our practical Wolfpack design, we exploit sensing diversity to provide sensing
confidence and apply it to sensing coverage. We have published our research results in [66] and address the following research challenges:

- Learning sensing diversity. We explore the accuracy and complexity of different machine learning techniques which can be used to learn the sensing capability of a deployment. We identify a method appropriate for resource constrained sensor networks which provides enough accuracy to meet user requirements but still requires low computation and communication overhead.

- On-demand sensor collaboration. Through learned sensing diversity, we determine when single sensors are sufficient to meet user requirements and when collaboration is necessary. When collaboration is needed, we determine the right sensors to collaborate to save valuable computation and communication resources.

- Distributed and online diversity exploitation. We provide a distributed solution to learning and exploiting sensing diversity, which allows for decreased bandwidth and energy usage and energy usage as well as increased scalability. Furthermore, our distributed scheme allows for increased adaptation to environmental dynamics since only portions of the network that do not meet user requirements are updated during runtime to ensure user requirements are met.

This chapter is organized as follows: We first explore sensing diversity in Section 4.1. We formally define our confident coverage problem with theoretical analysis in Section 4.2 and present our Wolfpack confident coverage design for practical system deployment in Section 4.3. We analyze the performance of Wolfpack in Section 4.4, and present conclusions and future work in Section 4.5.
4.1 Exploring and Exploiting Sensing Diversity

We explore how to take advantage of sensing diversity and on-demand sensor collaboration to provide confident sensing. We make use of the Wisconsin SensIT vehicle detection trace data [28], with 23 nodes deployed along a road with each node containing an acoustic, seismic, and infrared sensor. Vehicles make 20 passes along a road through the network with ground truth provided via a GPS trace. Trace data of raw sensor energy is provided for each sensor at a sampling rate of up to 4960Hz. We provide this unmodified real sensor data and ground truth as input to a trace-driven wireless sensor network simulation run on a PC. While the sensor data and ground truth is real, we simulate communication behavior and assume each node is a low power mote-class device equipped with an 802.15.4 radio, such as the Crossbow IRIS [24]. While we are aware that radio communication is often lossy in wireless sensor networks, we focus on sensing accuracy, not communication quality, and assume reliable communication.

We use the trace sensor data sampled at 100ms intervals and a total trace length of 6763 intervals. We classify sensor and sensor cluster readings into events when a vehicle is detected and non-events when no vehicle is present. To learn sensing diversity, collaborate sensors, and perform event detection, we use machine learning, which can address the complexity and heterogeneity of sensor data. We first identify sensing diversity among sensors of the same modality and among sensors of different modalities. Next, we illustrate the effects of sensing diversity on collaboration, determining when and how to collaborate sensors such that user requirements can be met. Finally, we compare two different machine learning techniques for learning sensing diversity, finding one appropriate for low power
4.1.1 Identifying Sensing Diversity

In this subsection, we use \( k \)-means clustering [12] with \( k = 2 \) classifications: trace data for each sensor is clustered into mean event and non-event centroids. As vehicles pass through the deployment area, each sensor reading is classified by determining its closest centroid. Using vehicle location ground truth and classified data, we plot the sensing range for all acoustic, seismic, and infrared sensors in Figure 4.1. Since sensing diversity encompasses sensing capabilities within a specific deployment, accounting for in-situ reality, some sensors have a sensing range of 0m, indicating that the vehicle does not pass close enough to the sensor to be detected.

![Figure 4.1: Sensing range differences in Wisconsin deployment.](image)

**Diversity within the same modality.** Figure 4.1 demonstrates that sensors of the same modality experience significant differences in event detection performance. For example, 10\% of acoustic sensors can detect vehicles at 400m, while another 10\% can only detect vehicles at ranges up to 50m. Similarly, 10\% of seismic sensors have a range of 40 meters or less while 10\% have a range greater than 200m. Similar differences can be observed for infrared sensors. This diverseness in sensing capability can be linked to the quality of the
sensor itself as well as the properties of the local environment such as terrain, weather, and other obstacles [56]. Due to sensing diversity, a single sensor may even perform differently in different environments. However, these observations are largely ignored in traditional sensing approaches. In [65], it is demonstrated that traditional sensing approaches, such as sensing coverage, use too little or too much sensing capability to detect events and either exhibit poor accuracy or waste energy. Other approaches such as [81] [133] also do not account for sensing diversity with respect to individual sensors, and thus fail to provide sufficient accuracy to meet user requirements.

Diversity among different modalities. Figure 4.1 also illustrates the differences between different sensing modalities. The sensing range of acoustic sensors is extremely varied, with 50% of acoustic sensors exhibiting a sensing range of at least 200m, with a minimum and maximum range of 0 and 400m, respectively. Conversely, infrared sensors have little variance, with 95% of sensors exhibiting a sensing range of 30m. Many existing detection coverage approaches [131] [13] rely on modality-specific sensing models, making sensor collaboration difficult in heterogeneous deployments. Therefore, significant sensing diversity exists in real deployments which should be addressed or exploited in confident sensing.

4.1.2 Impact of Diversity on Collaboration and Accuracy

In our study of sensor collaboration and accuracy, we compare Nearest Centroid, a variant of k-means clustering, with another learning technique, Fisher’s Linear Discriminant [12]. Both techniques allow for sensor readings to be combined into tuples for cluster-based collaboration and classification. In Nearest Centroid, sensor data is used to form two classi-
fication centroids, one for events and one for non-events, except that unlike $k$-means, ground truth is used in centroid formation. Like $k$-means, new data is classified by determining the closest centroid. While Nearest Centroid assumes that each reading in a tuple of sensor cluster data is independent, Fisher’s Linear Discriminant attempts to capture the dependencies among different sensors in a cluster. For example, sensors in proximity to each other usually yield correlated readings. Fisher’s Linear Discriminant attempts to find a projection (i.e. linear combination) of sensor cluster readings that maximizes the separation of event readings from non-event readings.

Using the vehicle detection trace, we select 103 detection locations with a 10m radius throughout the deployment area along a road on which vehicles pass. We detect vehicles at these target locations and classify such sampling intervals as events. For each target location, we form random clusters of size 2 through 25 with up to 30 clusters of each size, using sensors within 100m of each target location. We perform classification with both Nearest Centroid and Fisher’s Linear Discriminant and use ground truth for each individual sensor or cluster reading to determine accuracy. For each generated cluster we compare the detection accuracy of the best individual member sensor as a singleton cluster to the generated cluster accuracy and plot the results in Figures 4.2 and 4.3, where darker points indicate fewer than 50 overlapping clusters.

4.1.2.1 On-Demand Collaboration

We derive the guidelines for when sensor collaboration is needed and when it is not. First, we demonstrate that when an individual sensor meets the user detection requirements for a target location, collaboration is unnecessary. Using Figures 4.2 and 4.3, we show that in
over 36,000 cases for both Fisher's Linear Discriminant and Nearest Centroid, individual sensors have perfect accuracy. Individual sensors have over 95% accuracy in 2,000 more cases. In such cases where the user requirements fall in this 5% points accuracy range, collaboration is not needed for the requirements can be met by a single sensor and valuable communication and computational overhead is saved.

When an individual sensor cannot meet the user requirements for a given target location, collaboration is needed, but there are individual sensors that can be excluded from the collaboration process to reduce the search space size. In Figures 4.2 and 4.3, with this specific deployment and trace data, it is clear that no individual sensor can boost cluster accuracy as a cluster member by more than 20% points (we define this exclusion boundary as the sensitivity threshold). These sensors can be excluded from detection and collaboration, for any cluster consisting entirely of sensors below this threshold will not meet the user detection requirements.

Lastly, we show that when a sensor has a detection accuracy above the sensitivity threshold, but below the user requirements, it is a candidate for collaboration in a sensor
cluster. Figures 4.2 and 4.3 depicts over 2,500 cases where all individual sensors in a cluster exhibit less than perfect accuracy but the cluster accuracy is equal to 100%.

Table 4.1: Cluster vs. individual sensor performance.

<table>
<thead>
<tr>
<th>Cluster Type</th>
<th>Nearest Centroid</th>
<th>Fisher's Linear Discriminant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>3894</td>
<td>4606</td>
</tr>
<tr>
<td>Bad</td>
<td>1424</td>
<td>412</td>
</tr>
<tr>
<td>Neutral</td>
<td>36919</td>
<td>37219</td>
</tr>
</tbody>
</table>

4.1.2.2 Collaboration Accuracy and Complexity of Learning

Now that we have determined when sensor collaboration is required, we analyze how to learn the sensing capabilities of sensors and sensor clusters. We first show that when collaboration is needed, we must cluster sensors carefully instead of randomly. In Figures 4.2 and 4.3, cluster performance from randomly generated clusters can be classified into three categories: “good,” “bad,” and “neutral,” whose classification totals can be found in Table 4.1. Good clusters perform better than their best individual member sensors, bad clusters perform worse than their best individual member sensors, and neutral clusters perform the same as their best individual member sensors. It is clear that if the right sensors are not chosen for collaboration, cluster performance can suffer, for over 1,800 clusters exhibit worse performance than the best individual member sensor. However, clustering carefully can yield a cluster that performs accurately and better than the best individual member sensor.

We next compare two machine learning collaboration techniques, Nearest Centroid and Fisher’s Linear Discriminant and demonstrate that Fisher’s Linear Discriminant has better sensor collaboration performance. Figures 4.2 and 4.3 and Table 4.1 show that Fisher's
Linear Discriminant is able to eliminate most of the bad clusters and increase the number of good clusters since it accounts for the data dependencies among individual sensors in the cluster. While Fisher's Linear Discriminant improves clustering performance by eliminating many bad clusters, its greater computational requirements make it unattractive for use in sensor networks for distributed event detection, as most motes have microcontroller speeds of less than 10 MHz [24]. Given \( n \) training observations and a cluster size of \( k \), Nearest Centroid incurs a training cost of \( O(n \cdot k) \) while Fisher's Linear Discriminant incurs a much greater training cost of \( O(k^3 + nk^2) \). Although, from Figure 4.2, Nearest Centroid has many more "bad" clusters than Fisher's Linear Discriminant, its best performing clusters have comparable results to that of Fisher's Linear Discriminant. By collaborating sensors wisely, we can use Nearest Centroid to classify events in a distributed, node-centric fashion, achieving high performance and low complexity.

4.2 Diversity-Exploiting Confident Coverage

Confident sensing through exploitation of sensing diversity can apply to many problems such as assisted living [80] or military surveillance [121]; in this chapter we apply confident sensing to sensing coverage for vehicle detection in a static wireless sensor network. Here, we define our diversity-exploiting confident coverage problem and show that it is NP-hard. To provide confident coverage, a set of sensor clusters must be found that can meet user detection accuracy requirements for all desired detection locations in an energy conscious manner. We first define a set of nodes \( N = \{n_1, ..., n_n\} \). Each node \( n_j \in N \) contains \( k_j \) sensors, forming the set \( S \) of all sensors: \( S = \{s_1^1, s_2^1, ..., s_1^{k_1}, s_2^{k_1}, ..., s_2^{k_2}, ..., s_n^{k_n}\} \),
where \( s_j^i \) is the \( i \)th sensor on node \( j \). We also define a set of detection locations \( L = \{ l_1, \ldots, l_i \} \) which a user wishes to cover.

Users can specify the accuracy of detection for all locations in \( L \) in terms of desired false positive and false negative rates, \( ufp \) and \( ufn \), respectively. With machine learning, cluster \( C_i \) of one or more sensors can quantify its sensing diversity by determining its false positive rate and false negative rate for each location \( l_k \): \( fp(C_i, l_k) \) and \( fn(C_i, l_k) \). If cluster \( C_i \) meets the user requirements \( ufp \) and \( ufn \), for location \( l_k \), we say that location \( l_k \) is covered by \( C_i \).

A deployment has a set of possible sensor clusters \( C = \{ C_i | C_i \subseteq S \} \). With the learning techniques we have discussed, we can quantify cluster detection capabilities through a function \( f : C_i \rightarrow 2^L \) that maps a cluster \( C_i \in C \) to a subset of locations in \( L \) indicating the coverage of the locations by the sensors in \( C_i \). In this section, we assume that possible covering clusters are already generated and focus only on cluster selection for energy savings. In Section 4.3, we describe our combined cluster generation and selection process.

Our goal is to find the set of clusters \( C^* \subseteq C \) that meets the user requirements for all locations while residing on the fewest number of nodes, hence using the least amount of energy due to active node power consumption. We need to find a set of clusters \( C^* \subseteq C \) such that all locations in \( L \) are covered,

\[
\bigcup_{C_i \in C^*} f(C_i) = L
\]  

(subject to minimizing the total number of nodes contained by the clusters in \( C^* \):

\[
\text{minimize } |\{n_j \in N\}| \text{ where } s_j^k \in C_i \text{ for some } C_i \in C^* \text{ and some } k
\]  

(4.1)  

(4.2)
We now demonstrate that our cluster selection problem is NP-hard by showing that a special case of our problem is in fact the known NP-hard Set Cover problem [120]. In the special case, we assume that each node \( n \in N \) has only one sensor. That is, the set of all sensors in the deployment is represented as \( S = \{s_1, s_2, \ldots, s_n\} \), where a sensor \( s_i \) is the only sensor on node \( i \). We also assume that the set of possible clusters \( C \) contains only clusters with one sensor: \( C = \{\{s_i\}|s_i \in S\} \). Using the mapping of each cluster to covered locations, \( f : C_i \rightarrow 2^L \), then the set of all possible clusters is equivalent to: \( C = \{L'|L' \subseteq L\} \), where \( C \) represents a collection of subsets of \( L \). In this case, the optimization problem can be rewritten to find the set of clusters \( C^* \subseteq C \) such that:

\[
\bigcup_{L' \subseteq C^*} L' = L
\]  

(4.3)

subject to minimizing the number of clusters in \( C^* \). This special case is equivalent to the Set Cover problem, demonstrating that the general case of our clustering problem is also NP-hard. Since we wish to solve our confident coverage problem in a distributed manner, we plan to formally define a greedy solution and derive an approximation ratio in future work.

### 4.3 Wolfpack Framework Design

In this section, we propose Wolfpack, a practical, distributed solution to our confident coverage problem defined in Section 4.2. For each defined detection location, Wolfpack assigns a sensor cluster which meets the user detection requirements; these sensor clusters are formed in parallel at the start of deployment and updated when needed during runtime.
While many clustering schemes exist, such as leader election [43], Wolfpack clusters sensors on nodes based on their sensing capabilities and only clusters the sensors needed to meet the user requirements, placing unused nodes to sleep. A cluster is formed for each detection location by incrementally adding a member node and one or more of its sensors until the user requirements are met. Nodes are added to a cluster in decreasing order of the learned detection capability of their sensors. In some cases, a single sensor residing on a single node may be enough to meet the user requirements. We now describe the diversity aware clustering process in Section 4.3.1 and how clusters can be adaptively updated during runtime in Section 4.3.2 if they fail to meet user detection requirements.

4.3.1 Distributed Diversity-Aware Clustering

Nodes first quantify their sensing diversity and then compete to declare themselves as cluster heads, with the nodes that have the most sensing capability winning the competition. If a cluster formed solely from a cluster head node is not enough to meet the user requirements for its target location, the most capable member nodes join the cluster one at a time until the user requirements are met.

For event detection training and cluster formation, each active node maintains a history of recent observations for all of its sensors. Each node also maintains an application-level feedback mechanism, such as a vehicle tracking application, to provide event ground truth in a manner similar to [56]. Since we demonstrate in Section 4.1 that communication range of many sensor motes [24] is at least twice that of the sensing range, we cluster sensors within one communication hop of each detection location (fusion range) to save bandwidth and energy resources. We now describe the details of our distributed clustering scheme from
a node and event-driven perspective.

Exploring and quantifying diversity. Using Nearest Centroid, each node \( n_j \) explores its sensing diversity by training singleton clusters with each of its sensors \( s_j \) for each of the locations \( l_i \) in its fusion range. For each trained sensor and location, \( n_j \) determines the detection false positive and false negative rates \( fp(s_j, l_i) \) and \( fn(s_j, l_i) \). A sensor \( s_j \) is sensitive to location \( l_i \), or \( \text{sensitive}(s_j, l_i) \), if its false positive and false negative rates fall within 20% of the user requirements. This sensitivity threshold is an empirical rule that may vary with different deployment scenarios, for our choice of threshold is due to our study in Section 4.1 which shows that no individual sensor can improve detection accuracy by more than 20% when added to an existing sensor cluster.

Algorithm 4 Event Handler: Backoff Timer Fires

<table>
<thead>
<tr>
<th>Input</th>
<th>Node ( n_j ), sensitive locations ( L_s ) for node ( n_j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Cluster head or cluster member declaration for ( n_j )</td>
</tr>
<tr>
<td>1: for all ( l_i ) ( \in L_s ) do</td>
<td></td>
</tr>
<tr>
<td>2: if No cluster exists for ( l_i ) then</td>
<td></td>
</tr>
<tr>
<td>3: Create cluster ( C_i )</td>
<td></td>
</tr>
<tr>
<td>4: Set ( n_j ) as cluster head</td>
<td></td>
</tr>
<tr>
<td>5: Compute ( fp(C_i, l_i) ) and ( fn(C_i, l_i) )</td>
<td></td>
</tr>
<tr>
<td>6: else</td>
<td></td>
</tr>
<tr>
<td>7: Add ( n_j ) to existing cluster ( C_i )</td>
<td></td>
</tr>
<tr>
<td>8: end if</td>
<td></td>
</tr>
<tr>
<td>9: Broadcast the following in a packet:</td>
<td></td>
</tr>
<tr>
<td>10: Cluster ( C_i ) covers ( l_i ) with ( fp(C_i, l_i) ) and ( fn(C_i, l_i) )</td>
<td></td>
</tr>
<tr>
<td>11: if ( fp(C_i, l_i) &gt; upf ) or ( fn(C_i, l_i) &gt; ufn ) then</td>
<td></td>
</tr>
<tr>
<td>12: /<em>More collaboration is needed</em>/</td>
<td></td>
</tr>
<tr>
<td>13: Broadcast observation history for cluster ( C_i )</td>
<td></td>
</tr>
<tr>
<td>14: end if</td>
<td></td>
</tr>
<tr>
<td>15: end for</td>
<td></td>
</tr>
</tbody>
</table>

Each node then determines how much each of its sensors can contribute towards meeting the user requirements for each sensitive location: \( \Delta fp_j \) and \( \Delta fn_j \), which are real numbers between 0 and 1. Values closer to 0 indicate that the sensor \( s_j \) contributes very little towards meeting the user requirements for location \( l_i \), while the maximum possible values
1 - \(u_{fp}\) and 1 - \(u_{fn}\) indicate that the user requirements are met. \(\Delta f p_j\) is defined in Equation 4.4 (\(\Delta f n_j\) is similar).

\[
\Delta f p_j(s^m_j, l_i) = 1 - \max\{f p(s^m_j, l_i), u_{fp}\}
\]

A node quantifies its sensing diversity by calculating its importance, which is the sum of all contributions on a node for all of its sensors and sensitive locations. The more important a node, the more valuable it is towards meeting the user requirements for locations within its fusion range. More important nodes are more likely to have very capable sensors which will become members of many different clusters covering different detection locations. Each node sets a backoff timer based on its importance value to declare itself as a cluster head for its sensitive locations, where greater importance values result in shorter timers. Therefore, clusters can usually be formed from a small number of important nodes to cover all locations, thus reducing the number of active nodes needed to meet the user requirements. Importance is defined in Equation 4.5 as:

\[
I(n_j) = \sum_{m=1}^{\mid S_j \mid} \sum_{i=1}^{\mid L_j \mid} (\Delta f p_j(s^m_j, l_i) + \Delta f n_j(s^m_j, l_i))
\]

Importance-based competition. When a cluster head timer fires on node \(n_j\) (Algorithm 4), node \(n_j\) declares itself as the head for all sensitive locations not yet declared covered by another cluster head, creating a cluster \(C_i\) for each undeclared sensitive location \(l_i \in L_S\). If a cluster member timer fires on node \(n_j\), the node adds itself to an existing cluster \(C_i\) containing other member nodes. In both cases, the declaring node \(n_j\) adds to the cluster only its sensitive sensors that increase the \(\Delta f p_j\) and \(\Delta f n_j\) contribution towards
meeting the user requirements for each location $l_i$. The declared cluster $C_i$ is trained using Nearest Centroid learning and the observation history of all sensors in $C_i$.

**Algorithm 5 Event Handler: Receive Declaration Packet**

<table>
<thead>
<tr>
<th>Input</th>
<th>Node $n_j$, sensitive locations $L_S$ for node $n_j$, declaration cluster $C_i$ for location $l_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Node $n_j$ sets member timer if user requirements are not met for $C_i$</td>
</tr>
</tbody>
</table>

1: if $n_j \notin C_i$ and $l_i \in L_S$ then
2: if $n_j$ is competing to be a member covering $l_i$ then
3: Stop timer on node $n_j$ for $l_i$
4: end if
5: if $fp(C_i, l_i) > ufp$ or $fn(C_i, l_i) > ufn$ then
6: /* More collaboration is needed */
7: Update $I(n_j)$ using Eqns. 4.5 and 4.6
8: Set timer using $I(n_j)$; $n_j$ competes to join $C_i$
9: end if
10: end if

After a node $n_j$ has its backoff timer fire and it declares itself as a head or member, the node broadcasts a packet to all neighbors for each declared cluster $C_i$ with the cluster false positive and false negative rates, $fp(C_i, l_i)$ and $fn(C_i, l_i)$ and the location the cluster covers, $l_i$. If a declared cluster $C_i$ does not yet meet the user requirements $ufp$ or $ufn$, more collaboration is needed by recruiting member sensors, so node $n_j$ broadcasts its member sensor observation history to its neighbors to allow neighbors to compete to form a new cluster including observations from node $n_j$.

If a node $n_j$ receives a cluster head or cluster member declaration packet for an existing cluster $C_i$ and location $l_i$, node $n_j$ may perform one of two actions (Algorithm 5). First, if node $n_j$ is competing to become a member of cluster $C_i$, node $n_j$ has lost the member competition to the broadcasting node and cancels its backoff timer. Second, if the declared cluster $C_i$ does not meet the user requirements $ufp$ or $ufn$, and $n_j$ is sensitive to $l_i$, then $n_j$ attempts to add itself as a cluster member. Node $n_j$ determines its contribution towards meeting the requirements at $l_i$ if it adds itself as a member of cluster $C_i$. For each of its
sensitive sensors $s_j^n$, node $n_j$ updates $\Delta f_{p_j}$ and $\Delta f_{n_j}$ as performed in Equation 4.6 and sets an importance-based backoff timer to compete with other nodes to join $C_i$ based on Equation 4.5.

\[
\Delta f_{p_j}(C_i, s_j^n, l_i) = f_p(C_i, l_i) - \max\{f_{p_j}(C_i \cup \{s_j^n\}, l_i), f_{fp}\}
\]

(4.6)

**Example.** We present an example of the distributed clustering process in Figure 4.4. During initialization, in Figure 4.4 a), nodes $n_1$ and $n_2$ explore their diversity by determining which of their sensors are sensitive to each of the two locations, $l_1$ and $l_2$. From sensitivity and contributions towards meeting the user requirements $\Delta f_{p_j}$ and $\Delta f_{n_j}$, node $n_1$ quantifies its diversity, calculating an importance value greater than that of node $n_2$, thus setting a shorter backoff timer. With its shorter backoff timer, node $n_1$ declares itself as the cluster head for all its sensitive locations: $l_1$ and $l_2$ in Figure 4.4 b), but the cluster $C_2$ for location $l_2$ does not meet the user requirements. Node $n_2$ sets a backoff timer to join $C_2$, where its timer fires and in Figure 4.4 c) the new $C_2$ meets the user requirements.

![Figure 4.4: Distributed clustering with two nodes $n_1$, and $n_2$, each with two sensors, two detection locations $l_1$ and $l_2$, and user requirements $uf_p = uf_n = 0.05$. $n_1$ has greater sensing capability and importance: it becomes the cluster head for $C_1$ and $C_2$ (indicated by stars). $n_2$ is then added as a member to $C_2$ since the user requirements are not met by $n_1$ alone.](image-url)
4.3.2 Runtime and Adaptive Coverage

During runtime, after clusters have been formed to cover each location, cluster member nodes transmit sensor readings collected at each sample interval to their assigned cluster heads. Each cluster head makes a detection decision for each of its covered locations at each sample interval using its learned detection model and collected sensor data. To save bandwidth and transmission energy, we employ a scheme similar to [65], where a member node will only transmit sensor readings that are closest to the learned event centroid. A cluster head assumes the non-event centroid value if no transmission is received from a member sensor.

A cluster currently covering a location may experience a drop in performance, running the risk of not meeting the user detection requirements. Such a performance drop may be due to changes in background noise or to the properties of the event. In these cases, the existing cluster is dissolved and a new, more accurate cluster is formed in its place that meets the user requirements. Such an approach allows Wolfpack to adapt to environmental changes over time.

All cluster heads maintain moving accuracy using observation history and ground truth, allowing a cluster to detect a short term increase in $f_p(C_i, l_i)$ and $f_n(C_i, l_i)$. When such a performance drop is detected, the cluster head broadcasts an update message containing the location the cluster covers. Upon receiving the update message, all current cluster members stop transmitting samples to the cluster head. All nodes that have been awake long enough to have full observation histories compete to form a new cluster as in Section 4.3.1.

A new, updated cluster may change with respect to the old cluster in three ways. First,
the new cluster may consist of the same exact sensors as the old cluster, but with new event
and non-event centroids. Second, a newly formed cluster may also reside on the same nodes
as the old but contain different sensors. Third, a new cluster may also reside on different
nodes than the previous cluster. In future work, we will predict how a cluster changes during
an update, reducing energy and computational overhead in generating a new cluster.

For each cluster that is formed to cover a location, a subset of nodes is selected as
candidate nodes from among all sleeping and non-member sensors. Such candidate nodes
remain awake and sample data so that during an update candidates may be selected to
become member nodes if the current nodes cannot meet the user requirements.

4.4 Evaluation

We evaluate our scheme using the Wisconsin SensIT vehicle detection trace data [28], with
the same trace-driven simulation methodology described in Section 4.1. We choose 79
detection locations within the deployment area along the road where vehicles pass: the first
pass is used for initial training and the subsequent 10 passes are used for runtime detection.
The vehicle path deviates slightly with each pass, creating environmental dynamics during
runtime.

We compare our Wolfpack confident coverage approach to V-SAM [56], a state of the
art coverage scheme, which in contrast to conventional coverage approaches, attempts to
address sensing diversity by keeping awake sensors that sample similar data. For both
Wolfpack and V-SAM, we set the user requirements to 5% for both false positive and false
negative rates and set a 100m fusion range to collaborate sensors and to detect events at
each location. For Wolfpack, we use Nearest Centroid for clustering and collaboration if not explicitly specified. For comparison, we illustrate the performance of Wolfpack with and without adaptive coverage (AC) and V-SAM with both similarity coverage and forced k-coverage, where 1 to 3 nodes are awake to cover each location. We also test V-SAM with each possible k of n decision fusion rule: at least k of n awake nodes within the fusion range must detect an event in order for V-SAM to detect an event.

4.4.1 Meeting User Requirements

Table 4.2: Overall accuracy for Wolfpack and V-SAM.

<table>
<thead>
<tr>
<th></th>
<th>Acc. (%)</th>
<th>FP (%)</th>
<th>FN (%)</th>
<th>Loc. Met (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wolfpack</td>
<td>99.8</td>
<td>0.0</td>
<td>0.4</td>
<td>98.7</td>
</tr>
<tr>
<td>Wolfpack, No AC</td>
<td>99.2</td>
<td>0.0</td>
<td>1.9</td>
<td>93.7</td>
</tr>
<tr>
<td>Wolfpack, Fisher's</td>
<td>99.8</td>
<td>0.0</td>
<td>0.6</td>
<td>96.2</td>
</tr>
<tr>
<td>Wolfpack, No Collab.</td>
<td>99.5</td>
<td>0.0</td>
<td>1.4</td>
<td>88.6</td>
</tr>
<tr>
<td>V-SAM, Sim-cov, 1/n</td>
<td>93.4</td>
<td>0.0</td>
<td>15.1</td>
<td>67.1</td>
</tr>
<tr>
<td>V-SAM, 1-cov, 1/n</td>
<td>94.2</td>
<td>0.0</td>
<td>13.0</td>
<td>68.4</td>
</tr>
<tr>
<td>V-SAM, 2-cov, 1/n</td>
<td>94.6</td>
<td>1.6</td>
<td>9.7</td>
<td>74.7</td>
</tr>
<tr>
<td>V-SAM, 2-cov, 2/n</td>
<td>92.2</td>
<td>0.0</td>
<td>19.0</td>
<td>59.5</td>
</tr>
<tr>
<td>V-SAM, 3-cov, 1/n</td>
<td>94.8</td>
<td>2.2</td>
<td>8.0</td>
<td>57.0</td>
</tr>
<tr>
<td>V-SAM, 3-cov, 2/n</td>
<td>92.7</td>
<td>0.0</td>
<td>17.8</td>
<td>60.8</td>
</tr>
<tr>
<td>V-SAM, 3-cov, 3/n</td>
<td>84.6</td>
<td>0.0</td>
<td>39.8</td>
<td>31.6</td>
</tr>
</tbody>
</table>

First, we demonstrate that Wolfpack can meet the user requirements for nearly all detection locations, while V-SAM cannot. From Table 4.2, Wolfpack with Adaptive Coverage exhibits 99.8% accuracy and meets 98.7% of the detection locations while the best V-SAM configurations have 94.8% accuracy and meet only 74.7% locations. In analyzing performance by detection location, Figure 4.5 demonstrates that Wolfpack has perfect accuracy for 85% of locations while V-SAM has much higher variance, with 25% of locations ex-
hibiting less than 90% accuracy. In contrast with V-SAM, Wolfpack is able to quantify the sensing diversity in this deployment and choose the nodes with the most capable sensors to confidently detect events at each location. Conversely, V-SAM experiences poor accuracy, with false negative rates as high as 39.8% since it neither correctly learns the capabilities of each sensor nor collaborates carefully. V-SAM places too many nodes to sleep for lower $k$-coverage levels and fails to detect events, while increasing coverage levels only slightly increases accuracy.

![Location accuracy CDF](image)

**Figure 4.5:** Location accuracy CDF, illustrating the performance difference of each detection location.

Table 4.2 shows that Wolfpack with Adaptive Coverage is able to increase the locations covered by 5% points due to adaptation to environmental dynamics during runtime. Furthermore, the table demonstrates that by collaborating carefully and choosing different sensors, Nearest Centroid classification is able to slightly outperform Fisher's Linear Discriminant by covering 2.5% points more locations. The table also demonstrates that in most cases, single sensor clusters are enough to meet the user requirements, for when collaboration is disabled, the number of locations met decreases by about 10% points.
Table 4.3: Sensor, modality, and node makeup per cluster.

<table>
<thead>
<tr>
<th>No. Sensors</th>
<th>Acoustic</th>
<th>Seismic</th>
<th>Infrared</th>
<th>No. Nodes</th>
<th>No. Clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

4.4.2 Exploiting Sensing Diversity

Using Table 4.3, which depicts the node and sensor makeup of each cluster, we show that acoustic sensors are the most capable and are chosen in 80% of all clusters, with seismic and infrared sensors selected in 44% and 1% of all clusters. The table also illustrates that 64% of all clusters consist of a single sensor and 89% reside on a single node, emphasizing that Wolfpack only collaborates sensors when a single sensor cluster is not good enough to meet user requirements. When collaboration is disabled in Table 4.2, the relatively small 10% points reduction in locations that meet user requirements is explained by most clusters residing on a single node. Only 36% of all clusters consist of multiple sensors and 11% of all clusters consist of multiple nodes.

Figure 4.6: Cluster membership CDF of nodes and sensors for all detection locations.
In addition to exploiting sensing diversity among sensors of different modalities, Watchdog exploits sensing diversity among sensors of the same modality. Figure 4.6 shows that while nearly 60% of all nodes are not used to detect events at any location, one node is used to detect events at 46 different locations. Since each node has one sensor of each modality, some sensors of the same modality are much more capable than others, for Wolfpack heavily relies on a few nodes and sensors to confidently detect events. Similar behavior in cluster selection is witnessed for individual sensors.

4.4.3 Adaptive Coverage

![Figure 4.7: Location updates and accuracy during runtime.](image)

Using Figure 4.7, we demonstrate that Wolfpack with Adaptive Coverage is able to update and maintain accuracy when the environment changes, leading to an increase in locations covered by 5% points in Table 4.2. The figure shows that there are three distinct time periods where environmental dynamics cause an Adaptive Coverage update: intervals 50, 1100, and 3300. At these time intervals, the overall accuracy from Wolfpack with Adaptive Coverage maintains or drops only slightly while a greater decrease is witnessed for Wolfpack without Adaptive Coverage. Most clusters are able to tolerate moderate envi-
environmental dynamics and require no updates, as illustrated in Figure 4.8, with insignificant differences between Nearest Centroid and Fisher's Linear Discriminant. Over 10 locations experience only 1 update and only 1 location experiences 2 updates. Those locations that experience greater environmental changes experience lower detection accuracy: as the number of updates increases, accuracy drops from 100% to 95% with Nearest Centroid. Locations with significant environmental changes also incur cluster updates that move the cluster to different sensors or different nodes, as illustrated in Figure 4.9. As the complexity of the update increases, the accuracy decreases with Nearest Centroid from nearly 100% using the same sensors to 95% using different nodes. Fortunately, Figure 4.9 also shows that most updates form the same cluster (with different centroids) or use different sensors on the same nodes, maintaining higher accuracy than when new nodes are used.

4.4.4 Active Nodes and Energy Usage

By exploiting sensing diversity and selecting the nodes with the most capable sensors to cover each detection location, Wolfpack can avoid both too little and too much active node
coverage. In Table 4.4, all 3 Wolfpack configurations maintain 10 nodes awake at all times, while bested only slightly by V-SAM with similarity coverage. However, Wolfpack covers nearly all detection locations while similarity coverage V-SAM only covers 67%. Increasing levels of V-SAM coverage have more nodes awake, peaking at nearly 17 out of 23 total nodes awake. The large V-SAM standard deviation in Table 4.4 shows that V-SAM puts too many nodes to sleep to accurately capture all events, and once an event is captured, an unnecessarily large number of nodes are awoken to monitor the event. Consequently, due to fewer active nodes, Wolfpack uses less energy than V-SAM, illustrated by total energy usage in Table 4.4. For both schemes we measure energy usage as active node sampling time and transmission energy as defined in [110]. With only 10 nodes awake and very few communications due to most clusters residing on the same node, the most costly Wolfpack configuration, Nearest Centroid with Adaptive Coverage, uses 26.558J. This is compared with 32.933J for the most energy efficient V-SAM configuration, similarity coverage. As coverage is increased with V-SAM, energy usage also increases, for more nodes are awake, with energy usage nearly twice that of Wolfpack for 3-coverage V-SAM.

In Figure 4.10, we plot the CDF of energy usage per location for both Wolfpack and
Figure 4.10: CDF of energy usage per detection location for Wolfpack and V-SAM.

V-SAM. The figure demonstrates that for all locations, Wolfpack has very low energy consumption while for many locations, V-SAM can be very energy consuming. For all Wolfpack configurations, almost 90% of all locations are covered by a cluster that uses less than 4J of energy, with all locations using less than or equal to 5J. With similarity coverage, the most energy efficient V-SAM configuration, only 25% of locations use less than or equal to 5J, while 50% use more than 15J and others use nearly 20J.

4.5 Conclusion and Future Work

Existing work does not exploit sensing diversity in order to provide confident sensing. Thus, we explore how to use sensor collaboration to take advantage of sensing diversity. Through trace-driven study, we explore sensing diversity, when sensor collaboration is needed, and how to perform sensor collaboration. We formally define a diversity-exploiting confident coverage problem and demonstrate that it is NP-hard. For practical sensor network deployments, we also propose Wolfpack, a confident and distributed event detection coverage scheme which exploits sensing diversity to meet user detection requirements and save energy. Using real trace data for a vehicle detection application, Wolfpack outperforms existing ap-
proaches in terms of meeting user requirements, energy, and environmental adaptability. In future work, we plan to investigate how to duty cycle the most capable nodes and sensors to balance the energy consumption of the network while ensuring user requirements are still met.
Chapter 5

Practical Body Networking

The vast array of small wireless sensors is a boon to body sensor network (BSN) applications, especially in the context awareness and activity recognition arena. However, most activity recognition deployments and applications are challenged to provide personal control and practical functionality for everyday use. Such existing work provides no mobile on-body aggregator for sensor control and activity recognition feedback. Other works use a limited number of sensing modalities or use a separate classifier per modality which makes classification difficult for heterogeneous sensor deployments. Still more works do not adapt to the environmental dynamics present in BSNs or rely heavily on backend servers which can incur high communication overhead.

Towards providing a practical activity recognition solution, we present PBN: Practical Body Networking. Through the unification of TinyOS motes and Android smartphones, we combine the sensing power of on-body wireless sensors with the additional sensing power, computational resources, and user-friendly interface of an Android smartphone. We provide an accurate and efficient classification approach through the use of ensemble learning. We
explore the properties of different sensors and sensor data to further improve classification efficiency and reduce reliance on user annotated ground truth. We evaluate our PBN system with multiple subjects over a two week period and demonstrate that the system is easy to use, accurate, and appropriate for mobile devices. Our research results are published in [67], and in this chapter, we address the following challenges:

- A user-friendly solution. By combining the sensing power of TinyOS-based motes with the user interface, computation capability, and additional sensing capability of an Android smartphone, we provide a portable and easily configurable system with realtime feedback.

- Accurate classification. PBN handles both easy and difficult to classify activities, environmental changes, and noisy data. Our solution also allows context switching between activities without extensive parameter tuning and different, complex classifiers for each activity.

- Efficient classification. Since mobile hardware is constrained in terms of computation power and energy, we provide a sensor selection approach which only chooses the most helpful sensors for classification using a lightweight ensemble classifier.

- Reduced reliance on ground truth. Regular collection and labeling of training data can be invasive to the user, so we provide a retraining detection method that does not require ground truth information.

The rest of this chapter is organized as follows: Section 5.1 provides an overview of our PBN system design and Section 5.2 describes the ensemble classifier we use for activity
recognition. We describe our retraining detection method in Section 5.3, provide details on how to collect new training data in Section 5.4, and present a sensor selection method in Section 5.5. In Section 5.6, we evaluate our PBN platform and present conclusions and future work in Section 5.7.

5.1 System Overview and Architecture

In this section, we first present the application requirements. We then present our PBN hardware and application, which unifies TinyOS and Android. Next, we describe our PBN architecture and finally describe our PBN experimental setup which we refer to for the remainder of the chapter.

5.1.1 Application Requirements

Our PBN system design is motivated by the requirements of a practical body networking application for activity recognition. Data from multiple on-body sensors is reported to a mobile aggregator which makes classification decisions in real time. The system must be able to accurately and efficiently classify typical daily activities, postures, and environmental contexts, which we present in Table 5.1. Despite these categories being common for many individuals, previous work [78] has identified some of them, such as cleaning, as especially difficult to classify.

Table 5.1: PBN Classification Categories.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Indoors, Outdoors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Posture</td>
<td>Cycling, Lying Down, Sitting, Standing, Walking</td>
</tr>
<tr>
<td>Activity</td>
<td>Cleaning, Cycling, Driving, Eating, Meeting, Reading, Watching TV, Working</td>
</tr>
</tbody>
</table>
From the table, we break down our target classifications into three categories: Environment, Posture, and Activity. With the Environment and Posture categories, we can provide insight into the physical state of the user for personal health and physical fitness monitoring applications. We also wish to measure typical activities in which a user engages, such as watching TV, driving, meeting with colleagues, and cleaning. Here, we consider cycling and walking to be both postures and activities. Such activity recognition is quite useful for participatory sensing and social networking applications since it eliminates the need for a user to manually update his or her activities online. The requirements to provide such a practical activity recognition system are:

- **User-friendly.** Different on-body commercial off-the-shelf (COTS) sensor nodes must seamlessly interact with a mobile phone aggregator for simple user configuration and operation. The hardware must be portable and easy to wear, while the software must provide an intuitive interface for adding, removing, and configuring different sensors geared to detect the user's intended activities. Labeling training data should also be a simple and noninvasive task, facilitated by the mobile phone.

- **Accurate classification.** The system must accurately handle both easy and difficult to detect activities as well as noisy data and environmental dynamics. The system must also account for the changing geographic location of the user as well as the variable orientation of the individual on-body sensors.

- **Efficient classification.** A body sensor network for activity recognition may often use low power hardware, therefore, the classification algorithm should be computationally efficient in addressing the BSN dynamics of geographic location, user biomechan-
ics, and environmental noise. The system must also be energy efficient: by quantifying the contribution of each sensor towards accurately classifying activities, sensors with minimal contribution can be powered down. Furthermore, the system must avoid extensive parameter tuning as well as avoid unique, complex sensing models for each activity.

- **Less reliance on ground truth.** Activity recognition systems are often deployed with minimal labeled training data, thus the need exists to train a system in an online manner, requesting ground truth labels only when absolutely necessary. A reduced need for ground truth reduces the burden on the user to label training data.

## 5.1.2 PBN Hardware and Application

To achieve accurate and efficient activity recognition for mobile phones and on-body sensors, we provide an extensive hardware and software support system, which we describe in this section. In Section 5.1.2.1, we describe the implementation of USB host mode in the Android kernel to allow communication between a base station mote and an Android HTC phone. In Section 5.1.2.2, we describe our Android application for configuration and activity recognition.

### 5.1.2.1 Unification of Android and TinyOS

Our PBN system consists of Crossbow IRIS on-body sensor motes and a TelosB base station connected to an Android HTC G1 smartphone via USB. While previous work has connected TinyOS motes and mobile phones, such efforts either use energy demanding Bluetooth [32], do not provide mobile phone support for TinyOS packets [106], or use special hardware
Instead, we provide a seamless and efficient integration of TinyOS motes and Android smartphones. Our solution can be easily extended beyond the research-based TinyOS devices to work with a wide variety of commercial and more ergonomic USB and wireless sensors. Our challenges with such integration lie in 4 aspects: TinyOS sensing support, Android OS kernel support, Android hardware support, and TinyOS Java library support.

**TinyOS Sensing Support.** Illustrated in Figure 5.3, we implement a sensing application in TinyOS 2.x for Crossbow IRIS motes with attached MTS310 sensorboards. The sensor node application allows for runtime configuration of active sensors, sampling rates, and local aggregation methods. We develop a communication scheme to pass control and data packets through a TelosB base station connected to an Android smartphone.

**Android OS Kernel Support.** To prepare the phone to support external USB devices, a kernel EHCI or host controller driver is required. However, the USB host controller hardware documentation of the Google Developer phone is not publicly available. We incorporate the suggestions from [26] and modify the Freescale MPC5121 driver to work with the Qualcomm MSM7201A chipset on the Android phone. With these modifications, the host controller is able to recognize USB devices with a caveat: enabling the HOST controller disables the USB client mode for PC SD card access.

**Hardware Support.** The EHCI controller of the phone does not provide any power to external devices, such as the sensor motes in our case. To solve this limitation, we build a special USB cable that includes an external 5V battery pack with a peak load of 0.5A. The positive and ground cables are cut on the phone side such that only the USB device, not the phone, receives power. This also has the added benefit of not placing an extra load on the phone battery.
TinyOS Support on Android. Two challenges exist in providing Android TinyOS communication support: systems implementation and TinyOS software modifications. 1) Each mote has an FTDI USB serial chip that can be used to communicate with the host. The Linux FTDI driver creates a serial port device in the /dev directory. Android includes a minimal device manager that creates devices with insufficient privileges. This device manager is modified so that correct privileges are established. 2) TinyOS uses a binary serialization to communicate bidirectionally between the mote and host with the help of a C++ JNI interface. We modify the TinyOS JNI interface and associated Java libraries to compile and function on the Android platform. With such modifications, Android applications can send and receive TinyOS packets using the same Java interfaces available on a PC.

5.1.2.2 Android App

![Figure 5.1: PBN activity status view.](image)

![Figure 5.2: Ground truth logging.](image)

To provide a user-friendly front end for PBN, we implement an Android app to allow for easy configuration, runtime deployment, ground truth labeling, and data storage and
upload for offline analysis. The GUI component allows for user control of both phone and remote wireless sensors. The GUI also receives feedback as to the current activity and whether or not classifier retraining is needed, and also provides ground truth labels to the classifier.

**Sensor Configuration.** Our PBN Android app provides an interface to allow for easy configuration of both phone sensors as well as remote TinyOS sensors. Using our software interface, a user can easily add or remove TinyOS nodes as well as phone and TinyOS-based sensors. The user is also able to adjust sampling rates as well as local data aggregation intervals to reduce the number of radio transmissions. A user's sensor and sampling configuration can be stored on the phone in an XML file so that configuration must only be performed once. Users of different phones can also exchange saved sensor configurations.

**Runtime Control and Feedback.** With a created sensor configuration, a PBN user is able to quickly start and stop data sampling and activity recognition. As depicted in Figure 5.1, the current activity is displayed during runtime along with a configurable histogram of the most recent activities. When our classifier determines that accuracy is dropping and retraining with more labeled ground truth is needed, the PBN app prompts the user to input the current activity, illustrated in Figure 5.2. During retraining, an indicator and button on the current activity screen appears, allowing the user to log any changes in the ground truth. Labeled training data can be stored locally for later retraining or uploaded to a server for sharing and offline analysis.
5.1.3 PBN Architecture

Illustrated in Figure 5.3, our Practical Body Networking (PBN) system resides solely on TinyOS-based motes and on an Android smartphone with no reliance on a backend server. Multiple TinyOS-based motes, each containing one or more sensors of different modalities, are attached on-body. Each mote communicates wirelessly (dotted lines) with a TinyOS mote base station, which is connected via USB (solid lines) to an Android smartphone.

Phone and mote sensors (white dotted areas) sample data at a user configured rate and aggregate data from each sensor over a larger aggregation interval. Aggregated data for each sensor is returned at each aggregation interval. For each TinyOS mote, aggregated data is returned wirelessly in a single packet. We provide a reliable communication scheme between the phone and motes and determine through our 2 week experiment that for most activities, packet loss rates are below 10%, with 99% of packets received after one retransmission, even
when using the lowest transmission power.

Aggregated data from phone and mote sensors is fed into the PBN classification system (gray dotted area in Figure 5.3) at each aggregation interval to make a classification decision. The classifier, AdaBoost, is initially trained with the user labeling a short two minute period of each pre-defined activity (Ground Truth Management) but training can be updated online through Retraining Detection. During initial training and retraining, the Sensor Selection module reduces the training overhead incurred by AdaBoost by choosing only the most capable sensors for performing accurate classification. We now describe the core of our PBN platform:

**Activity Classification.** We use AdaBoost [36], an ensemble learning algorithm, as our activity recognition classifier which resides entirely on a smartphone. AdaBoost combines an ensemble of weak classifiers together to form a single, more robust classifier. With this approach, we are able to train weak classifiers for each sensor in our deployment and combine them together to recognize activities. AdaBoost is able to maximize training accuracy by selecting only the most capable sensors for use during runtime. We improve upon AdaBoost by providing online training and retraining detection as well as improve computational overhead through sensor selection.

**Retraining Detection.** Since initial training data may not be sufficient to account for body sensor network dynamics, AdaBoost retraining is needed in order to ensure high accuracy throughout the deployment lifetime. We investigate the discriminative power of individual sensors and from our analysis we are able to detect when retraining is needed during runtime without the use of labeled ground truth. For each sensor, we use Kullback-Leibler divergence [74] to determine when runtime data is sufficiently different from training
data and use a consensus-based approach to initiate retraining when enough sensors indicate that retraining is needed.

**Ground Truth Management.** When retraining is needed, we investigate how much new data to collect and label in order to ensure BSN dynamics are captured, yet minimize the intrusiveness of the user manually annotating his or her current activities. We also determine how to maintain a balance of training data for each activity to ensure AdaBoost trains properly and provides maximum runtime accuracy.

**Sensor Selection.** During training, AdaBoost trains multiple weak classifiers for every sensor in the deployment, even if many sensors are never chosen by AdaBoost when training is complete. Through analysis of the theory behind ensemble learning, we identify both helpful and redundant sensors through the use of the Pearson correlation coefficient [107]. Based on our analysis, we provide a method to give AdaBoost only the sensors that provide a significant contribution towards maximizing accuracy, thus reducing online training overhead.

### 5.1.4 Experimental Setup

For the remainder of the chapter, we will refer to our activity recognition experiment, in which we collected two weeks of sensor data using two subjects, depicted in Figures 5.4 and 5.5. Each subject wore five Crossbow IRIS motes wirelessly linked to a TelosB base station and Android HTC G1 smartphone. The mote and sensor configuration for our experiment is summarized in Table 6.1. On the phone, which we attach to the waist, we make use of the 3-axis accelerometer as well as velocity from WiFi and GPS, with GPS active only when PBN detects the user is outdoors. On the mote, we use an MTS310 sensorboard with the
following sensors: 2-axis accelerometer, microphone, light, and temperature. In addition to the sensors on the mote, the base station also collects RSSI information from each received packet, which has been previously demonstrated [100] to provide insight into body posture. During initial and online training, all sensors are active, while only sensors selected during training remain active during the remaining sampling periods.

**Table 5.2: PBN Deployment Configuration.**

<table>
<thead>
<tr>
<th>Node</th>
<th>ID</th>
<th>Location</th>
<th>Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone</td>
<td>0</td>
<td>R. Waist</td>
<td>3-Axis Acc., GPS/WiFi (velocity)</td>
</tr>
<tr>
<td>IRIS</td>
<td>1</td>
<td>L. Wrist</td>
<td>2-Axis Acc., Mic., Light, Temp.</td>
</tr>
<tr>
<td>IRIS</td>
<td>2</td>
<td>R. Wrist</td>
<td>2-Axis Acc., Mic., Light, Temp.</td>
</tr>
<tr>
<td>IRIS</td>
<td>3</td>
<td>L. Ankle</td>
<td>2-Axis Acc., Mic., Light, Temp.</td>
</tr>
<tr>
<td>IRIS</td>
<td>5</td>
<td>Head</td>
<td>2-Axis Acc., Mic., Light, Temp.</td>
</tr>
</tbody>
</table>

For the microphones and accelerometers, raw ADC values are sampled at 20ms intervals to ensure quick body movements can be captured, with light and temperature ADC readings sampled at 1s intervals, and GPS/WiFi sampled every 10s. To reduce communication
overhead, data for each sensor is aggregated locally on each node at 10s intervals, which is well within the time granularity of the activities we classify. To reduce the complexity of local aggregation, data from each accelerometer axis is treated as a separate sensor. During local aggregation, light and temperature sensor readings are averaged since these sensor readings remain relatively stable for each activity. Except for GPS/WiFi, all other sensors compute the difference between the highest and lowest readings for each aggregation interval, for the change in readings indicate body movement or sound.

During the two week period, both subjects recorded all activity ground truth in order to evaluate the accuracy of training data (training accuracy) and runtime accuracy. We recorded ground truth for 3 classification categories, illustrated in Table 5.1: Environment, Posture, and Activity.

5.2 AdaBoost Activity Recognition

The core of our activity recognition approach uses ensemble learning, specifically AdaBoost.M2 [36], which we expand and improve upon in subsequent sections. In this section, we explain how we adapt AdaBoost to run on a phone for use with a body sensor network. AdaBoost is lightweight enough for mobile phones, yet previous work [78] [94], while relying on offline processing with no feedback or user control, has demonstrated AdaBoost to be accurate for classification applications. Furthermore, without user parameter tuning, our implementation of AdaBoost is able to choose the right sensors needed to maximize training accuracy for all activities. Other classifiers commonly used for activity recognition use a combination of complex, specialized classifiers per sensor modality [87] [90] or per activity
which require extensive parameter tuning. Other techniques use single classifiers which are computationally demanding for mobile phones, such as GMMs [89] or HMMs [39].

Using AdaBoost, we incrementally build an ensemble of computationally inexpensive weak classifiers, each of which is trained from the labeled training observations of a single sensor. Weak classifiers need only to make classification decisions that are slightly correlated with the ground truth; their capabilities are combined to form a single accurate classifier. The completed ensemble may contain multiple weak classifiers for the same sensor; some sensors may not have trained classifiers in the ensemble at all. AdaBoost incrementally creates such sensor-based weak classifiers by emphasizing the training observations misclassified by previous classifiers, thus ensuring that training accuracy is maximized.

Using Algorithm 6, we describe AdaBoost training. We define a set of activities $A = \{a_1, \ldots, a_n\}$, sensors $S = \{s_1, \ldots, s_m\}$, and observation vectors $O_j$ for each sensor $s_j \in S$, where each sensor has $n$ training observations. The training output is an ensemble of weak classifiers $H = \{h_1, \ldots, h_T\}$, where $h_t \in H$ represents the weak classifier chosen in the $t^{th}$ iteration. We initialize a set of equal weights $D_1$ for each training observation, where during the training process, greater weights for an observation represent greater classification difficulty.

During each iteration $t$, we train a weak classifier $h_{t,j}$ for each sensor $s_j \in S$ using observations $O_j$ and weights $D_t$. We then compute the weighted classifier error $e_{t,j}$ for each trained sensor classifier, adding only the sensor classifier to $H$ which has the lowest weighted error. Before the next iteration, the observation weights $D_t$ are updated based on the current weights and the misclassifications made by the selected classifier.

Given an observation $o$, each weak classifier returns a probability vector $[0, 1]^a$ with each
Algorithm 6 AdaBoost Training

Input  Max iterations $T$, training obs. vector $O_j$ for each sensor $s_j \in S$, obs. ground truth labels
Output  Set of weak classifiers $H$

1: Initialize observation weights $D_1$ to $1/n$ for all obs.
2: for $t = 1$ to $T$ do
3:    for sensor $s_j \in S$ do
4:        Train weak classifier $h_{t,j}$ using obs. $O_j$, weights $D_t$
5:        Get weighted error $\epsilon_{t,j}$ for $h_{t,j}$ using labels
6:    end for
7:    Add the $h_{t,j}$ with least error $\epsilon_t$ to $H$ by choosing $h_{t,j}$ with least $\epsilon_t$
8:    Set $D_{t+1}$ using $D_t$, misclassifications made by $h_t$ [36]
9: end for

A scalar representing the probability that the current activity is $a_i$. To train a weak classifier $h_{t,j}$ for each sensor $s_j \in S$, we use a naive Bayes model where training observations $O_j$ are placed into one of 10 discrete bins. The bin interval is based on the minimum and maximum possible readings for each sensor. Each binned training observation from $O_j$ is assigned its respective weight from $D_t$ and ground truth label in the set of activities $A$. A new observation is classified by placing it into a bin, with the generated probability output vector corresponding to the weights of training observations present for each activity in the assigned bin. With a weak classifier chosen for each iteration, Equation 5.1 defines the output of the AdaBoost classifier for each new observation $o$ during runtime:

$$h(o) = \arg\max_{a_i \in A} \sum_{t=1}^{T} \left( \log \frac{1 - \epsilon_t}{\epsilon_t} \right) h_t(o,a_i)$$ (5.1)

In Equation 5.1, the activity probability vector for each weak classifier $h_t$ is weighted by the inverse of its error $\epsilon_t$. Thus, the weak classifiers with the lowest training error have the most weight in making classification decisions. To put it another way, AdaBoost chooses the sensors with weak classifiers that minimize weighted training error, achieving maximum training accuracy for all activities.
In Figure 5.6, we depict AdaBoost training and runtime performance for $T = 1$ to 300 AdaBoost iterations using data from Subject 1 and ground truth from the Activity classification category. For this chapter, we use 300 iterations to achieve maximum accuracy. We also show in Figure 5.6 that AdaBoost does not choose all of the 34 sensors in our experiment, for even with 300 iterations it only chooses 18. During runtime, any sensor not selected by AdaBoost will be disabled to save energy and communication costs.

Lastly, in Figure 5.7, we demonstrate that AdaBoost can perform very accurately with a sizable amount of training data: maximum training and runtime accuracy is achieved with roughly 50 observations per activity for Subject 1. However, through retraining in Section 5.3 we can train AdaBoost with a very small set of initial training data and update AdaBoost during runtime.

### 5.3 Retraining Detection

In this section, we propose an online training approach to achieve high accuracy with a limited initial training data set. This approach can also be used to retrain when an existing
data set is not accurate enough. First, we investigate how to quantify the discriminative power of each sensor to detect when runtime data is significantly different than training data. Second, we use the insight we gain from the discrimination analysis to detect when AdaBoost retraining is needed during runtime without retrieval of ground truth information.

5.3.1 Sensor Data Discrimination Analysis

We investigate how to quantify the discriminative power of each sensor and predict if it is accurate, employing the use of Kullback-Leibler divergence [74]. K-L divergence measures the expected amount of information required to transform samples from a distribution $P$ into a second distribution $Q$. Using trace data from Subject 1, we demonstrate the use of K-L divergence per activity to identify which sensors are best able to distinguish between activities. We also show a clear relationship between K-L divergence per activity and training accuracy. We conclude that K-L divergence can be used to detect when retraining is needed without regular ground truth requests to compute classifier accuracy: sensors need only to compare training data to current runtime data.

![Figure 5.8: K-L divergence for each sensor and each activity.](image)

In Figure 5.8, we analyze the ability of each sensor to discriminate between different activities; we calculate K-L divergence as “one against the rest,” for one data distribution
is calculated for the target activity and another distribution is calculated for all other activities. For all analyses using K-L divergence, we discretize continuous-valued sensor data into 100 bins. The figure shows that some sensors, such as those on the hands (nodes 1 and 2) have poor ability to distinguish between any activity. Conversely, sensors on the feet (nodes 3 and 4) are especially good at detecting activities that involve motion, such as walking and cycling. The GPS/WiFi velocity (0-LOC) has the highest K-L divergence of all for detecting driving, with a value of nearly 14.

![Figure 5.9](image)

**Figure 5.9:** Sensor K-L divergence and training accuracy.

In Figure 5.9, we compare K-L divergence per sensor and activity to individual sensor training accuracy using Nearest Centroid [66]. Nearest Centroid is a variant of k-means clustering and unlike the AdaBoost weak classifiers, it is an inexpensive classifier that does not require a weight for each training observation. The figure shows a clear relationship between K-L divergence and runtime accuracy, indicating that for a given activity, sensors with high K-L divergence are much more likely to have high accuracy compared with sensors with low K-L divergence. As with Figure 5.8, activities involving motion, such as cycling, walking, and driving are most easily distinguished and have the highest accuracy. From Figures 5.8 and 5.9, we can conclude that K-L divergence will allow sensors to tell activities apart and can even be used to tell if training data and runtime data for the same activity
are sufficiently different that retraining is needed.

5.3.2 Consensus-based Detection

During runtime, retraining detection is performed in two steps, using the insight gained in Section 5.3.1. First, at each aggregation interval, each active sensor independently determines if retraining is needed. Second, for some aggregation interval, if enough sensors determine that retraining is needed, PBN prompts the user to record ground truth for a short period. During this period, all sensors are woken up and sample data which is then labeled by the user. When the retraining period completes, a new AdaBoost model is trained using both the old and new training data.

**Step 1. Individual Sensor Retraining Detection.** In Section 5.3.1, we demonstrate in that K-L divergence is a powerful tool for determining sensor classification capability. Here, we use the discriminative power of K-L divergence to determine when retraining is needed for each sensor. For a sensor to determine retraining is needed, two conditions must hold: 1) The runtime data distribution for the current activity must be significantly different from the training data distribution, and 2) The runtime data distribution must have at least as many observations as the training data distribution.

During training, each sensor chosen by AdaBoost computes K-L divergence for each activity using its labeled sensor training data. Training K-L divergence per activity is used as a baseline to compare against during runtime to determine when retraining is needed. For each activity $a_i \in A$, training sensor data distribution $T_i$ for activity $a_i$, and training sensor data distribution $T_o$ for all activities $a_j \in A \setminus \{a_i\}$, each sensor computes the K-L divergence between $T_i$ and $T_o$, $D_{KL}(T_i, T_o)$. Then, during runtime, for each new observation, AdaBoost
classifies the observation as activity $a_i \in A$, and each active sensor adds its data to a runtime distribution $R_i$ for activity $a_i$. An active sensor $s_j$ determines retraining is needed when the runtime-training K-L divergence is greater than the training K-L divergence for the current activity $a_i$: $D_{KL}(R_i, T_i) > D_{KL}(T_i, T_o)$.

During runtime, to ensure a fair comparison between training and runtime K-L divergence, each sensor does not determine if retraining is needed until the runtime distribution $R_i$ has as at least as many observations as the training data distribution $T_i$. We determine through evaluation that collecting fewer runtime observations per activity yields similar accuracy but more retraining instances, which imparts a greater burden on the user. Since Figure 5.7 demonstrates that 100 observations per activity is more than sufficient to achieve maximum runtime accuracy (this is also true for Subject 2), we limit training data to 100 observations per activity to reduce delay before the training observations and runtime observations are compared.

**Step 2. Consensus Detection: Combining Individual Sensor Decisions.** During each runtime interval, PBN checks to see how many sensors indicate retraining is necessary. If a weighted number of sensors surpasses a threshold, new ground truth is collected and a new AdaBoost classifier is trained. We describe how this consensus threshold for retraining is obtained.

First, upon completion of AdaBoost training, we can determine the AdaBoost weight of the sensor classifiers used to make correct classification decisions. Through an extension of Equation 5.1, in Equation 5.2, we can output the weight of the correct weak classifiers given an observation $o$ and correct AdaBoost decision $a_i$, $w(o, a_i)$:
\[ w(o, a_t) = \sum_{t=1}^{T} \left( \log \frac{1 - \epsilon_t}{\epsilon_t} \right) h_t(o, a_t) \] (5.2)

Next, using training data, a trained AdaBoost classifier, and Equation 5.2, we determine the average weight \( w_{\text{avg}} \) for all correct training classification decisions \((o, a_t)\). To do this, we compute a weight \( w_j \) for each active sensor \( s_j \), indicating how important its decisions are relative to other sensors. Depicted in Equation 5.3, \( w_j \) is computed as the sum of the inverse error rates for each weak classifier belonging to sensor \( s_j \) (multiple classifiers for \( s_j \) may be iteratively trained and chosen by AdaBoost). Function \( f(j) \) maps sensor \( s_j \) to each AdaBoost iteration \( t \) where AdaBoost chooses the weak classifier trained by \( s_j \).

\[ w_j = \sum_{\forall t \in f(j)} \log \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \] (5.3)

At each runtime interval, we sum the weights \( w_j \) for each sensor \( s_j \) that indicates retraining is necessary. If the sum of the sensor weights is greater than \( w_{\text{avg}} \), PBN notifies the user to collect ground truth and retrain a new AdaBoost classifier.

5.4 Ground Truth Management

We address two issues regarding labeling and addition of new sensor data to the existing training dataset during retraining. First, we address how much new data to add to the training set. Second, we show how to maintain a balance between training data set sizes for each activity to ensure AdaBoost re-trains properly and maintains high runtime accuracy.

When PBN decides retraining is required, it will prompt the user to log ground truth for a window of \( N \) aggregation intervals. Through evaluation, we find that recording ground
truth retroactively for the data that triggered the retraining results in no change in accuracy since any such significant change in sensor data is persistent. During the ground truth logging period, a user only notes the current activity at the start of the logging period and any activity changes throughout the remainder of the logging period. In the evaluation, we determine that with $N = 30$ (5 minutes), the retraining ground truth collection window is short enough not to be intrusive to the user but long enough to capture changes in PBN dynamics. When the ground truth labeling period is complete, the new labeled sensor data is added to the existing training set and a new AdaBoost model is trained.

Since the core of AdaBoost training relies on creating a weight distribution for all training observations based on classification difficulty, each activity must be given nearly equal amounts of training data (within an order of magnitude) for AdaBoost to train properly [46]. Without a balance in training observations across all activities, AdaBoost will focus on training activities with more training observations, creating poor runtime accuracy for activities with fewer training observations. However, if we are too restrictive in enforcing such a balance, very few new training observations will be added to the training dataset during retraining, resulting in poor adaptation to the new data. In Equation 5.4, we ensure that each activity $a_i \in A$ has no more than $\delta$ times the average number of training observations per activity, where $O$ is the set of training observations.

$$\frac{|O_i| - \frac{1}{|A|} \sum_{\forall O_k \in A} |O_k|}{\frac{1}{|A|} \sum_{\forall O_k \in A} |O_k|} \leq \delta$$

The limit imposed in Equation 5.4 allows AdaBoost to place equal emphasis on training weak classifiers for each activity during retraining. If, during retraining, an activity exceeds this limit, data is removed until the number of observations is under the limit. We remove
observations at random to reach the activity limit since we do not know how representative each observation is of its labeled activity; some observations may contain more noise than others.

5.5 Sensor Selection for Efficient Classification

While AdaBoost, through training, provides a measure of sensor selection by weighting the most accurate sensors, this approach can be computationally demanding: at each AdaBoost training iteration, a weak classifier is trained and evaluated for each sensor. In this section, we focus on identifying redundant sensors and excluding them from AdaBoost training to reduce the number of weak classifiers trained by AdaBoost. To achieve this goal, we first investigate why ensemble learning algorithms are successful: weak classifiers must be relatively accurate and different weak classifiers must have diverse prediction results [142]. Second, by identifying sensors that satisfy these properties, we provide a method to detect redundant sensors during runtime and exclude them from input to AdaBoost during online retraining. Our method adapts to BSN dynamics during runtime to ensure only helpful sensors are used by AdaBoost.

5.5.1 Identifying Sensing Redundancies

With ensemble learning and AdaBoost, each weak classifier in the ensemble must be slightly correlated with the true classification to achieve high runtime accuracy. Since we use data from a single sensor to train each weak classifier, it follows that sensors chosen by AdaBoost will be similarly correlated both in terms of raw data and in classification decisions made by each weak classifier. While the Pearson correlation coefficient [107] is also used in previous
works [115] [143] to identify packet loss relationships between different wireless radio links; here we use correlation to identify sensing relationships between different sensors to identify both sensors that are helpful as well as redundant.

Figure 5.10 depicts the correlation between each sensor pair using the raw sensor data collected from Subject 1. It is noted that several correlation patterns exist: accelerometers are strongly correlated as are light and temperature sensors. We can use this information to find sensors with redundant data and remove them from the AdaBoost training process to save computational overhead as well as energy consumption.

We next illustrate that not only do correlation patterns exist between raw sensor data but also between sensor and sensor cluster classifier decisions. In Figure 5.11, with data from Subject 1, we show that when we add a new sensor to an existing sensor cluster, the greatest accuracy increase is achieved when the sensor and cluster have uncorrelated classification decisions. The figure shows the decision correlation and accuracy change for
Figure 5.11: Decision correlation between individual sensors and sensor clusters.

nearly 7,000 randomly generated clusters from size 1 through 19 using data from Subject 1, with individual sensor and sensor cluster classifiers trained using Nearest Centroid [66].

To compute the decision correlation for a classifier, each correct decision is recorded as 1 and each incorrect decision is recorded as 0. From the figure, it is clear that choosing sensors with a decision correlation close to 0 can help select sensors that will make the most contribution towards an accurate sensor cluster.

5.5.2 Adaptive Runtime Sensor Selection

We now describe how to reduce the number of sensors given as input to AdaBoost during online retraining, reducing retraining overhead while achieving high runtime accuracy. We perform sensor selection using raw sensor data correlation, also extending the concept to sensor and sensor cluster classifier decision correlation.

Sensor selection consists of two components: Threshold Adjustment and Selection. During Threshold Adjustment, using a trained AdaBoost classifier, a threshold $\alpha$ is computed which discriminates between sensors chosen by AdaBoost and unused sensors.
tion, a previously computed $\alpha$ value is used to select a set of sensors for input to AdaBoost during retraining. Threshold Adjustment is performed during initial training while Selection is performed during subsequent retrainings. To ensure the selection threshold stays current with BSN dynamics, we update the threshold $\alpha$ periodically during runtime retraining using Threshold Adjustment and apply smoothing using a moving window of previous $\alpha$ values.

**Algorithm 7 Raw Correlation Threshold for Sensor Selection**

<table>
<thead>
<tr>
<th>Input</th>
<th>Set of sensors $S$ selected by AdaBoost, training observations for all sensors $O$, multiplier $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Sensor selection threshold $\alpha$</td>
</tr>
</tbody>
</table>

1. $R = \emptyset$ // set of correlation coefficients
2. for all combinations of sensors $s_i$ and $s_j$ in $S$ do
3. Compute correlation coefficient $r = |r_{O_i,O_j}|$
4. $R = R \cup \{r\}$
5. end for
6. // compute threshold as avg + (n * std. dev.) of $R$
7. $\alpha = \mu_R + n\sigma_R$

**Threshold Adjustment.** During initial training, we initialize a sensor selection threshold $\alpha$ and update it during runtime retraining to adjust to any changes in user geographical location, biomechanics, or environmental noise. To determine the threshold $\alpha$, we first train AdaBoost with all sensors as input and determine $S$, the set of sensors selected by AdaBoost. Using Algorithm 7, we determine the correlation coefficient $r$ of raw sensor training data for each combination of sensor pairs in $S$. With each correlation coefficient $r$ stored in set $R$ for all sensors selected by AdaBoost, we determine the mean correlation coefficient $\mu_R$ and standard deviation $\sigma_R$. We then set the threshold $\alpha$ to be $n$ times the standard deviation above the mean correlation. We determine through empirical evaluation that $n = 2$ is sufficient to include nearly all sensors selected by AdaBoost but exclude unused sensors.

**Selection.** During retraining, when the threshold is not being updated, we choose a set of sensors $S^*$ from the set of all sensors $S$ using the previously computed threshold $\alpha$. The
selected set $S^*$ is given as input to AdaBoost to reduce the overhead incurred by training on the entire set $S$. In Algorithm 8, we ensure that no two sensors in $S^*$ have a correlation coefficient that is greater than or equal to the threshold $\alpha$, since we demonstrate previously in Section 5.5.1 that correlation closest to 0 yields the most accurate sensor clusters. To enforce the threshold, we maintain a set $E$, which contains sensors that have a correlation coefficient above the threshold for some sensor already in $S^*$. For each sensor pair in $S$, we determine the correlation coefficient $r$ and add pairs to $S^*$ where $r < \alpha$. When $r \geq \alpha$ for some sensor pair, we add the less accurate sensor to $E$ as determined by Nearest Centroid [66] and the other to $S^*$ as long as it is not already in $E$.

**Algorithm 8** Sensor Selection Using Raw Correlation

```plaintext
Input Set of all sensors $S$, training observations for all sensors $O$, threshold $\alpha$

Output Selected sensors $S^*$ to give as input to AdaBoost

1: $S^* = \emptyset$
2: $E = \emptyset$ // set of sensors we exclude
3: for all combinations of sensors $s_i$ and $s_j$ in $S$ do
4:   Compute correlation coefficient $r = |r_{o_i, o_j}|$
5:   if $r < \alpha$ then
6:     if $s_i \notin E$ then $S^* = S^* \cup \{s_i\}$
7:     if $s_j \notin E$ then $S^* = S^* \cup \{s_j\}$
8:   else if $r \geq \alpha$ and acc$(s_i) >$ acc$(s_j)$ then
9:     // use accuracy to decide which to add to $S^*$
10:    if $s_i \notin E$ then $S^* = S^* \cup \{s_i\}$
11:    $E = E \cup \{s_j\}$; $S^* = S^* \setminus \{s_j\}$
12:   else
13:     if $s_j \notin E$ then $S^* = S^* \cup \{s_j\}$
14:     $E = E \cup \{s_i\}$; $S^* = S^* \setminus \{s_i\}$
15:   end if
16: end for
```

**Pair DC.** We also propose two other correlation metrics with which to select sensors: pair decision correlation (Pair DC) and cluster decision correlation (Cluster DC). With these two metrics, we select sensors based on classification decisions made by individual sensors and sensor clusters. For a sensor or cluster classifier trained using Nearest Centroid, we
can convert each training data decision into an integer value as in Section 5.5.1 to use in computing decision correlation. Sensor selection using sensor pair decision correlation is identical to raw data correlation, except for the use of individual sensor classifier decisions rather than raw data.

**Cluster DC.** For sensor cluster decision correlation, we modify Algorithms 7 and 8. Instead of iterating through all sensor pair combinations, we first find an individual sensor classifier with the highest accuracy using Nearest Centroid and add it to a trained sensor cluster $C^*$. We then incrementally add sensors to $C^*$ as long as the sensor has a decision correlation with the cluster that is below the threshold. The final cluster is then used by AdaBoost for training.

### 5.6 Evaluation

We evaluate our PBN activity recognition system using the configuration and data collection methods described in Section 5.1.4, using two weeks of activity recognition data and two subjects, one of whom is not an author. While these two subjects have substantially different routines and compose very different evaluation scenarios, we leave a more broadly focused evaluation with more subjects to future work. We first evaluate classification performance with a good initial training dataset in Section 5.6.1 and show that PBN can achieve good accuracy for even difficult to classify activities. In Section 5.6.2, we evaluate online training with a limited initial training dataset and compare our PBN retraining detection method to a periodic retraining approach, illustrating the benefits of PBN retraining detection. We then show the effectiveness of our sensor selection approach in Section 5.6.3 and evaluate
our PBN application performance in terms of mobile phone hardware constraints in Section 5.6.4.

### 5.6.1 Classification Performance

Figure 5.12: Runtime performance comparison for multiple subjects and multiple classification category sets. Subject 2 does not engage in the cycling, lying down, cleaning, and reading categories, so the corresponding bars are not presented.

In Figure 5.12, we first compare performance for both subjects in the three classification categories depicted in Table 5.1: Environment, Posture, and Activity, showing that PBN is able to achieve high runtime accuracy with a good initial training dataset (100 observations per activity) along with no online training or sensor selection. Subject 2 does not perform the cycling, lying down, cleaning, or reading categories, hence there is no histogram bar. In
addition to total accuracy, we plot accuracy, precision (true positive/(true positive + false positive)), and recall (true positive/(true positive + false negative) for each activity.

Each classification category in Figure 5.12 has similar performance characteristics per subject: Subject 1 has total accuracies of 98%, 85%, and 90% for the Environmental, Posture, and Activity categories, while Subject 2 has respective total accuracies of 81%, 82%, and 76%. Interestingly, Subject 1 performs significantly better than Subject 2 since Subject 2 often performs each activity less cleanly, for example, working with different lights on in a room or eating in different locations, sometimes with friends. Lastly, more complex activities, which involve a combination of different physical movements as well as external sounds and light, perform reasonably well in comparison with more easily classified activities. Subject 1 has over 90% accuracy for complex activities like cleaning, eating, meeting, reading, and watching TV. Subject 2 also has over 80% accuracy for complex activities like eating, meeting, and watching TV.

![Figure 5.13: PBN decision and ground truth timeline for Subject 1.](image)

We present a classification timeline for Subject 1 using the Activity category in Figure 5.13, comparing PBN classifications with ground truth. Here, it is apparent that activities involving movement, such as cycling, walking, and driving, are classified with few errors. More misclassifications are seen for more complex activities, such as working confused with
meeting, cleaning, and watching TV, however, as illustrated in Figure 5.12, overall accuracy for each of these complex activities is near or above 90%.

![Figure 5.14: AdaBoost sensor weights per activity.](image)

In Figure 5.14, we show the normalized AdaBoost weights for each sensor for Subject 1 and the Activity classification category as calculated in Equation 5.3. In addition to the weights for the total of all classifications, we also compute the normalized weight of correct decisions made by each sensor for each activity using runtime data. This figure indicates that AdaBoost is able to select the right sensors to maximize training and runtime accuracy and exclude 16 unhelpful sensors (shown in black). The figure also shows heavy reliance on the light and temperature sensors to distinguish between indoor and outdoor activities as well as reliance on the accelerometers to detect activities involving motion, such as walking, driving, and cycling.

### 5.6.2 Online Training

In this section, we demonstrate that we can use a small training dataset, perform online training through retraining detection, and achieve similar accuracy as if we had a larger initial training data set. In this section, we focus on the Activity classification category and also initialize each runtime configuration with 10 random training data samples for
each activity. To limit AdaBoost training overhead, we enforce a maximum of 100 training observations per activity since Figure 5.7 demonstrates that this number is more than enough to achieve maximum runtime accuracy. Since we are using random initial training data, we compute the average and standard deviation for 10 runs of each configuration.

**Ground Truth Management.** First, we investigate the retraining window size described in Section 5.4 for collecting new ground truth in Figure 5.15. From the figure, larger window sizes mean significantly less retraining and computational overhead but also less runtime accuracy. We use a smaller window size of 30 new data elements per retraining to balance accuracy and computational overhead. We argue that roughly 20-40 retraining instances per subject over a two week period as per Figure 5.15 is an inconsequential burden, since the subject must only interact with the phone once per instance to input his or her current activity.

Next, we show the ideal training data balance restriction $\delta$, from Equation 5.4, in Figure 5.16. As $\delta$ approaches 2.0, runtime accuracy increases and the number of retraining instances decreases. We choose a $\delta = 2.0$ to achieve high accuracy and fewer retraining instances, as larger $\delta$ values do not improve accuracy or reduce the number of retraining instances.
Furthermore, since we enforce a maximum number of training observations per activity, larger $\delta$ values will have no effect on the balance of training data among activities.

**Comparison with Periodic Retraining.** In Figure 5.17, we compare the best retraining performance with our retraining detection and ground truth management methods to a naive retraining approach: periodic retraining. We implement periodic retraining for periods of 100 to 500 intervals, with each retraining adding 30 new training data elements to the training data set. We also choose $\delta = 2.0$ for training data balance among all activities. The figure demonstrates that PBN with retraining detection achieves high accuracy for both subjects while periodic retraining either has twice as many retraining instances for the same accuracy or especially in the case of Subject 2, lower accuracy for fewer retraining instances.

We also show that PBN with retraining detection achieves similar accuracy as periodic retraining but incurs fewer retraining instances. In Figure 5.18, we present a timeline of runtime accuracy and retraining instances for the first 2500 classification intervals comparing PBN to periodic retraining every 100 classification intervals. Both approaches have very volatile initial accuracy, but eventually converge by 2000 intervals. The figures also
Figure 5.18: Runtime timeline with accuracy and retraining instances.

demonstrate that by reducing the number of retraining instances, PBN retraining detection consequently reduces the amount of ground truth logging by the user.

5.6.3 Sensor Selection

We now evaluate our sensor selection approach in Section 5.5 and demonstrate that we can exclude 30-40% of all sensors from AdaBoost training, yet achieve similar runtime accuracy as using no sensor selection. Removing this many sensors from training is a significant savings in computational overhead since these sensors no longer have classifiers trained during each of 300 AdaBoost training iterations.

Figure 5.19: Sensor selection comparison with online training.
Using 10 random observations per activity as initial training data and using online training, we compare each of the correlation metrics to determine which is the best in terms of accuracy and sensors excluded. In Figure 5.19, we depict the percentage of total sensors excluded from AdaBoost (SS Only), the percentage of total sensors excluded by sensor selection and AdaBoost training (SS + AdaBoost), and average runtime accuracy for each sensor selection method.

Figure 5.19 indicates that both sensor pair raw correlation and decision correlation exclude 30-40% of all sensors from training. Both of these methods also have nearly the same accuracy as without sensor selection, indicating that the right sensors are excluded using these methods. While sensor cluster sensor selection excludes 50% of sensors from training for Subject 1, accuracy is also worse. The figure also shows that in combination with AdaBoost training, each sensor selection method excludes more than 10% points more sensors than with no sensor selection, resulting in additional energy savings since unused nodes are powered down. We suggest that using raw correlation is the best approach to sensor selection, as it requires the least computational overhead of the three metrics, yet excludes a significant number of sensors and maintains high runtime accuracy.

5.6.4 Application Performance

During our experiments, the wireless motes had battery life measured in days (4 or 5 days of 8 hour sessions), while the Android HTC G1 phone was unable to last more than 8 hours without recharging. Here, we focus on evaluating the phone performance, for its battery lifetime is much shorter. In Table 5.3, we measure the CPU, memory, and power consumption of our PBN application to demonstrate that it is practical for mobile phones.
<table>
<thead>
<tr>
<th>Mode</th>
<th>CPU</th>
<th>Memory</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle (No PBN)</td>
<td>&lt;1%</td>
<td>4.30MB</td>
<td>360.59mW</td>
</tr>
<tr>
<td>Sampling (WiFi)</td>
<td>19%</td>
<td>8.16MB</td>
<td>517.74mW</td>
</tr>
<tr>
<td>Sampling (GPS)</td>
<td>21%</td>
<td>8.47MB</td>
<td>711.74mW</td>
</tr>
<tr>
<td>Sampling (WiFi) + Train</td>
<td>100%</td>
<td>9.48MB</td>
<td>601.02mW</td>
</tr>
<tr>
<td>Sampling (WiFi) + Classify</td>
<td>21%</td>
<td>9.45MB</td>
<td>513.57mW</td>
</tr>
</tbody>
</table>

Table 5.3: PBN CPU, memory, and power benchmarks.

We run each configuration for 5 minutes and show the average for system idle, sampling only with GPS or WiFi for localization, sampling with AdaBoost training, and sampling with AdaBoost classification.

Since retraining occurs infrequently, during the vast majority of system deployment (Sampling + Classify), PBN incurs roughly 20% CPU use with memory overhead under 10MB. Most of this overhead is due to the TinyOS Java libraries sending and receiving packets to and from sensor motes; we leave it to future work to further optimize these libraries for mobile devices.

When retraining does occur, it takes between 1 and 10 minutes on the HTC G1, depending on training data size. With newer hardware (HTC Nexus One), retraining time is halved under the same conditions. Since PBN retraining is run as a background process, it can be preempted and has little impact on performance of other applications, such as checking email or making a phone call.

Depicted in Figure 5.20, we evaluate PBN power consumption with the display off using a power meter from Monsoon Technologies. In Table 5.3, we demonstrate that during sampling and classification, PBN consumes roughly 150mW in addition to system idle.
This consumption is about 1/3 of the additional 450mW consumed by the display when it is active. GPS-based localization consumes an additional 200mW, however GPS is enabled only when WiFi localization is not possible. An additional 90mW is consumed during online training, however, as previously mentioned, these periods are short lived and infrequent.

5.7 Conclusion and Future Work

In this chapter, we present PBN, a significant effort towards a practical solution for daily activity recognition. PBN provides a user-friendly experience for the wearer as well as strong classification performance. Through integration of Android and TinyOS, we provide a software and hardware support system which couples the sensing power of on-body wireless sensors with an easy to use mobile phone application. Our approach is computationally appropriate for mobile phones and wireless motes and also chooses the sensors that maximize accuracy. We improve AdaBoost through online training and enhanced sensor selection, analyzing the properties of sensors and sensor data to identify helpful and redundant sensors as well as indicators for when retraining is needed. We show in our evaluation that PBN can perform accurately for classifying a wide array of activities and postures even when using a limited training dataset coupled with online training. In future work, we intend to provide an extensive usability study with a diverse array of subjects as well as improve energy use on the phone.
Chapter 6

Remora

Specialized personal sensing applications, especially in the context awareness and activity recognition domain, are ideally suited for body sensor network (BSN) deployments. Specifically, the wide variety of sensor modalities available for on-body nodes provide sensing capability that far exceeds using smartphones alone. Additionally, the use of a smartphone in conjunction with external on-body nodes provides additional sensing power, computational capability, portability, and a user-friendly interface for personal control and runtime feedback. Activity recognition applications that can exploit such a system include assisted living [130], physical fitness assessment [2], and patient monitoring [83] [20]. A physician may administer BSNs for retirement community residents [130] [103] to detect depression, ensuring proper eating, social activity, and exercise. Similarly, a university sports team coach may deploy BSNs on his or her student-athletes to ensure optimal performance [6]. The BSN worn by each student-athlete can not only measure athletic performance but also detect daily living habits that may be detrimental, such as excessive social activity or lack of studying.
Smartphone-based BSN applications which use activity recognition to assess daily living habits, such as those mentioned above, demand high classification accuracy and long system lifetimes. However, many individual BSNs may exhibit poor accuracy due to specific user behavior, background noise, and even difficult to classify activities. For example, an activity classifier may be easily confused between a meeting with colleagues and watching television. Furthermore, smartphone batteries are quickly drained after 8-10 hours of BSN use [67], thus requiring frequent recharges.

Interestingly, users of many BSN applications may be in close proximity to one another, belonging to groups with strong interpersonal ties. Many residents of a retirement community are close friends and engage in many activities together. Athletes on a university sports team will not only practice together but live, eat, study, and party together. In our motivational study and evaluation, we find that subjects spend between 20-50% of waking hours in the proximity of a close friend, family member, or colleague.

Consequently, we propose that BSNs in physical proximity to one another opportunistically share resources. In this paper, we focus on activity recognition and share neighboring resources to extend device lifetimes and increase activity classification accuracy. BSN neighbors, such as family and friends, exploit overheard on-body sensor data transmissions to increase classification accuracy. Increased accuracy by sharing is possible due to available neighbor sensors that are both individually accurate and have complimentary classification capabilities. Through sharing, neighbors can use fewer sensors, allowing more to be disabled to save energy. Furthermore, to increase phone battery life, classifiers are duty cycled among neighbors, allowing the phone to go into a low power sleep.

However, three prominent challenges arise from sharing resources among neighboring
BSNs. First, we must determine when sharing provides an energy benefit and when it does not. We must accurately characterize sharing costs and benefits as well as predict when neighboring BSNs will be together long enough to achieve such benefits. Second, we must identify the cases where sharing improves accuracy, such as difficult to classify activities, then find and utilize the resources that provide the best combination of accuracy and energy savings. Lastly, we must provide a lightweight and flexible sharing approach to limit sharing overhead and adapt to the dynamics of available neighbors. This sharing approach must provide an activity classification method which efficiently addresses changes in sensor availability over time. Neighboring BSNs must also easily collaborate to decide which sensors and classifiers to share.

Most existing approaches to activity classification ignore sharing altogether, whether using smartphones [86] [87] or on-body sensors [67] [109]. Other approaches rely extensively on backend servers [8] [76] [35] for classifier training and dissemination of classifiers to phones. One effort [89] shares classifiers and classification results among neighbors but the energy costs and benefits are not fully addressed.

Towards addressing the above challenges and shortcomings, we show through an initial experiment how sharing sensors among neighboring BSNs can increase activity classification accuracy and save sensor energy by using fewer sensors. The insights gained from this experiment motivate the design of our system, Remora. Remora is an opportunistic resource sharing approach which improves classification accuracy and extends system lifetime among BSNs in proximity to one another. With Remora, we first determine the costs and benefits of sharing: we determine energy overhead as well as the proximity duration needed for the sharing energy benefit to outweigh the energy costs. Next, we provide a sharing-aware
classification approach which uses an ensemble classifier that efficiently adapts to changes in neighbor and sensor availability. This approach allows sharing BSNs to jointly select sensors to maximize training accuracy and use as few sensors as possible to save sensor energy. To save phone energy, sharing BSNs only use one active classifier per time period. Our main contributions are:

- We analyze the overhead of sharing sensors and classifiers with a time and energy model, only sharing when neighboring BSNs will be together long enough for sharing to benefit.

- We provide an efficient method to share sensors and classifiers among neighboring BSNs. A collaborative approach allows neighbors to share only a small set of accurate and complimentary sensors and duty cycle classifiers to save phone energy.

- With two weeks of evaluation from six subjects, in comparison with using only individual BSN resources, Remora can increase activity classification accuracy by up to 30% and extend battery lifetime by over 65%.

This chapter is organized as follows: We explore the feasibility of sharing and present a motivational study in Section 6.1. Using our motivational experiment, we present our Remora design and example applications in Section 6.2. In Section 6.3, we discuss how BSNs detect neighbors and analyze sharing costs and benefits. We present our Sharing-Aware Classification approach in Section 6.4, performance evaluation is given in Section 6.5, and present conclusions and future work in Section 6.6.
6.1 Feasibility and Motivation

In this section, we discuss the intuition behind our Remora BSN resource sharing system. We first define our activity recognition application, hardware platform, and design goals. We next discuss the feasibility of sharing and how sensors and classifiers are shared among BSNs. Then, in a short experiment, we show the potential accuracy and energy benefits of sharing.

Problem Statement. In this paper, we provide a personal activity recognition application and target personal activities such as walking, working at a desk, having a meeting with colleagues, driving a car, or watching TV. To classify activities, we use smartphone-based body sensor networks, combining on-body wireless sensors with the additional sensing capability, computational resources, and user interface of a smartphone. Using BSN hardware, our goal is to provide an activity recognition system which allows neighboring BSNs to share resources in order to increase activity classification accuracy while saving phone and on-body sensor energy.

6.1.1 Feasibility

We address three issues that affect BSN resource sharing and its success in increasing classification accuracy and providing energy savings. We first demonstrate that in real scenarios, there is enough opportunity for BSN neighbors to share and provide a benefit. Second, we describe available sensing and classification resources and how best to share them among neighbors. Lastly, we address privacy concerns.

Sharing Opportunity. Our approach targets applications where the users have strong
interpersonal ties. In such scenarios, there is sufficient interaction between neighboring BSNs such that sharing can provide a significant impact on overall classification performance and energy savings. In the MIT Reality Mining dataset [31], physical interactions of nearly 100 subjects were recorded using Bluetooth phones over 9 months. Most subjects were friends or colleagues: students and faculty who worked in the same building and spent time together off campus. Analysis of the MIT data yields that, on average, 25% of the time each subject is in proximity with at least one other subject. Our evaluation in Section 6.5 yields similar results: for each subject, 30-50% of the time was spent in proximity with at least one neighbor.

**Figure 6.1:** Sharing Hierarchy. We focus on the proximity layers, sharing classifiers and sensors among neighbors.

**Sharing Hierarchy.** In Figure 6.1, we present a hierarchy of BSN resources which are eligible for sharing among users. We use the bottom two (Sensor and Classifier) to exploit proximity, improving accuracy and energy use. In the Sensor Layer, when 2 or more BSNs are in proximity to each other, the phone for each BSN overhears the transmissions of the others' sensor nodes. Neighboring BSNs freeride, opportunistically using the overheard data directly to train their own classifiers and make activity classification decisions. Through
collaboration, neighbors select a set of sensors that achieves higher accuracy and uses fewer combined sensors compared with individual classification.

Previous work demonstrates that continuously active classification can drain a phone battery in as little as 8 hours [67], so extending phone lifetime is critical. At the Classifier Layer, neighbors duty cycle classifiers so that at any given time, only one active classifier is running, allowing all other phones to go into a low power sleep state. Since the active classifier makes classification decisions for all neighbors, neighbors only share if they are all performing the same activity. However, as demonstrated by our evaluation, neighbors in close proximity are likely to be performing the same activity. Additionally, a short duty cycle time allows quick detection of activity changes while sharing.

Most existing efforts for sensing resource sharing perform at the top level, or Cloud Layer, where shared classifiers are trained on a backend server [89] or data is relayed [25] among different users. However, most cloud-level systems rely on expensive classification algorithms, may incur high communication overhead to transfer sensor and classifier information, and also do not fully exploit proximity.

Privacy. We provide several features to address privacy concerns. First, previous work has established that people are more likely to share with others in close physical proximity [127], such as friends and colleagues. Because users share only with neighbors in physical proximity, sharing neighbors are already able to visually identify the activities being performed. Second, a user can define "private" sensors which are not shared while "public" sensors are shared and broadcast data to neighbors. For example, a user may refuse to share a wireless heart rate monitor or pulse oximeter, defining such sensors as private. Furthermore, sensors such as these may not help neighbor classification accuracy.
Third, each on-body node aggregates sensor data samples before transmitting, providing coarse-grained aggregated data to both the local and neighboring BSNs. A similar technique [25] is used to obfuscate personally identifiable characteristics for phone sensor data.

6.1.2 Motivation

To motivate our Remora design, we present a pilot study which demonstrates that sharing resources among BSNs yields significant accuracy improvement and energy savings. We first detail our experimental configuration, which we extend upon in the evaluation in Section 6.5. Then, we show results from our motivation experiment in which two subjects wear BSNs simultaneously, performing shared activities.

6.1.2.1 Experimental Configuration

![BSN configuration](image)

**Figure 6.2**: BSN configuration: 4 on-body wireless sensor nodes communicate with a base station node which is attached to an Android phone.

Each subject in our experiments wore four TinyOS-based Crossbow IRIS motes, shown
in Figure 6.2. Each mote is wirelessly linked to a TelosB base station, which is connected via USB to an Android HTC Hero smartphone. Our solution can be extended beyond the research-based TinyOS devices to work with more ergonomic commercial sensors. We present details on sensors, sampling, and classification:

<table>
<thead>
<tr>
<th>Node</th>
<th>Location</th>
<th>Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone</td>
<td>R. Waist</td>
<td>3-Axis Acc., GPS/WiFi (velocity)</td>
</tr>
<tr>
<td>IRIS</td>
<td>L. Wrist</td>
<td>2-Axis Acc., Mic., Light, Temp.</td>
</tr>
<tr>
<td>IRIS</td>
<td>R. Wrist</td>
<td>2-Axis Acc., Mic., Light, Temp.</td>
</tr>
<tr>
<td>IRIS</td>
<td>L. Ankle</td>
<td>2-Axis Acc., Mic., Light, Temp.</td>
</tr>
<tr>
<td>IRIS</td>
<td>R. Ankle</td>
<td>2-Axis Acc., Mic., Light, Temp.</td>
</tr>
</tbody>
</table>

**Table 6.1**: Deployment Configuration.

**Sensors.** The sensor configuration is summarized in Table 6.1. On the phone, which we attach to the waist, we make use of the 3-axis accelerometer as well as velocity from WiFi and GPS, with GPS active only when WiFi is unavailable. On the mote, we use an MTS310 sensorboard with the following sensors: 2-axis accelerometer, microphone, light, and temperature. In addition to the sensors on the mote, the base station also collects RSSI information from each received packet, which has been previously demonstrated [100] to provide insight into body posture. Each subject makes all on-body sensors public (shared with all neighbors) and all phone sensors private (used by the local BSN only).

**Sampling and Aggregation.** For the microphones and accelerometers, raw ADC values are sampled at 20ms intervals to ensure quick body movements can be captured, with light and temperature ADC readings sampled at 1s intervals, and GPS/WiFi sampled every 10s. To reduce communication overhead, data for each sensor is aggregated locally on each node at 10s intervals, which is well within the time granularity of the activities we classify. During local aggregation, light and temperature sensor readings are averaged
since these sensor readings remain relatively stable for each activity. Except for GPS/WiFi, all other sensors compute the difference between the highest and lowest readings for each aggregation interval, for the change in readings indicate body movement or sound.

Aggregated data for all sensors on a mote is combined into a single packet and broadcasted to the local phone and any neighboring phones. Motes transmit at the lowest available sending power to save energy and reduce congestion while a reliable communication scheme with the local phone eliminates packet loss with fewer than 1% retransmissions.

**Classification.** At each aggregation interval, aggregated data is used to classify activities with a Bootstrap Aggregating (Bagging) [15] classifier, detailed in Section 6.4. During the experiment, subjects recorded all activity ground truth in order to evaluate the accuracy of training data (training accuracy) and runtime accuracy.

### 6.1.2.2 Identifying Sharing Benefits

Through a shared activity experiment with 2 BSNs, we show how sharing can improve accuracy and provide energy savings. Two subjects performed four shared activities (driving,
reading, walking, and watching TV) for over four hours in length. We use the same data to compute both individual and shared classification results, using 10 initial observations per activity as training data. Since Bagging trains nondeterministically, we plot average runtime accuracy and standard deviation over 30 runs in Figure 6.3, demonstrating stable performance. From the figure, when both BSNs share each other's sensors, this results in a total accuracy increase of 12% points for Subject 1 and 5% points for Subject 2. This is because the reading and watching TV activities are performed in the same room and are often confused when only individual sensors are available. However, due to their different locations, sensors from a neighboring BSN provide complimentary information and can be exploited to provide higher accuracy for both activities.

In Figure 6.4, we compare the accuracy of sensors at different body locations. The figure shows that on-body sensors improve accuracy significantly compared with using phone sensors only. For both the individual and shared scenarios, accuracy is improved by over 25% when using all available on-body sensors. Leg sensors give the greatest boost, for they remain still during sitting activities and in motion while walking, which is easily captured by accelerometers.

![Figure 6.5: Randomly selected sensor clusters.](image-url)
Lastly, we show that we can save energy by choosing only the most capable sensors and turning off unneeded sensors. For Subject 1, we generate 100 random sensor clusters of sizes 10 through 40 from all available sensors, including public sensors on Subject 2. Training classifiers for each cluster, we plot the minimum, maximum, and average accuracy in Figure 6.5. The figure shows that if we only choose 10 sensors, we can still achieve 97% accuracy, as long as we choose the right 10 sensors. This result motivates us to provide an efficient sensor selection approach for shared BSNs, described in Section 6.4.2, that chooses a small number of sensors to achieve both high accuracy and node energy savings.

6.2 Architecture and Applications

![Remora Architecture](image)

Figure 6.6: Remora Architecture. Neighbor Management determines if sharing with detected neighbors will provide an energy benefit. Sharing-Aware Classification collaborates with neighbors to select sensors for shared classification, classifies sensor data from the local phone, local motes, and neighboring motes, and duty cycles classifiers among neighboring phones.

With our goal of energy and accuracy gains through BSN resource sharing, we present
the Remora system architecture in Figure 6.6. Each BSN consists of TinyOS-based motes and an Android smartphone with no reliance on a backend server. For each BSN, multiple on-body motes (Local Motes) communicate wirelessly with the phone (Local Phone). While our system uses a USB-connected base station as an 802.15.4 relay between other phones and motes, Remora can also use other communication modalities, such as Bluetooth.

During runtime, Neighbor Management detects neighbors and initiates sharing with neighbors only if sharing will provide an energy benefit. Sharing-Aware Classification trains classifiers and classifies activities using local sensors as well as neighbor sensors made available by Neighbor Management. Sharing-Aware Classification also duty cycles classifiers among sharing BSNs to save phone energy. We now describe the core of our Remora system, with our significant contributions highlighted in gray in Figure 6.6:

**Neighbor Management.** The Neighbor Management module characterizes the costs and benefits of sharing and initiates sharing only when an energy benefit is possible. Proximity Detection and Duration Prediction detects neighboring BSNs and estimates how long detected neighbors will be in proximity. When a neighbor is detected, Cost-Benefit Analysis determines the energy costs and benefits of sharing using an empirically generated time and power model as well as the proximity duration estimate. Sharing is initiated when Cost-Benefit Analysis determines that a neighbor will be in proximity long enough for the energy benefit of sharing to exceed any additional energy cost to collect new ground truth and train a shared classifier.

**Sharing-Aware Classification.** The Sharing-Aware Classification module provides a classification and training approach which adapts to the dynamics of available neighbors, utilizing neighbor and local resources which provide the best combination of high accuracy
and energy savings. To efficiently classify activities on the phone in the presence of changing neighbor availability, we use Bagging [15], an ensemble classifier. Bagging allows a Remora BSN to quickly create an accurate classifier by combining weak classifiers from available local and neighbor sensors. At each aggregation interval, a decision classified by the ensemble is pushed to the application as well as pulled by other neighbors whose phones recently returned from a low power sleep state.

Collaborative Sensor Selection allows neighboring BSNs to find and utilize the shared resources which, in comparison with individual classification, provides higher accuracy and greater sensor energy savings. At the start of runtime, each BSN uses a classifier for individual classification based on available training data (Training Data Management) or a classifier previously trained through Collaborative Sensor Selection. When sharing is initiated, Collaborative Sensor Selection first labels training data with user input if no training data is present for the neighboring sensors. BSNs then collaborate to choose only the most capable sensors, creating a Bagging classifier that achieves high training accuracy for all neighbors yet uses fewer on-body sensors. Unused sensors are disabled during runtime to save energy.

To save phone energy, sharing BSNs duty cycle classifiers (Duty Cycling). One active neighbor makes a classification decision for the group at each aggregation interval. For each inactive BSN, as long as users are not interacting with the phone, the phone enters a low power sleep state to save energy. After sharing is initiated and training is complete, neighbors collaborate to establish a duty cycle order. Duty cycle times are short enough (5 min. in evaluation) so that changes in neighbor dynamics can be captured. Sleeping phones can be woken by the user to poll the active classifier for the current activity. Since
duty cycling classifiers allows an active BSN to make decisions for all sleeping neighbors, all neighbors must be performing the same activity. However, neighbors performing different activities can still share sensors to increase accuracy. In our evaluation, subjects rarely perform different activities together for long durations, so our design focuses on both sensor and classifier sharing and we leave classifying different simultaneous activities to future work.

**Figure 6.7:** Activity status.  
**Figure 6.8:** Shared resources.  
**Figure 6.9:** Map view of individual and shared activities.

**UI and Applications.** To provide a user-friendly front end for Remora, we implement an Android app to allow easy configuration, ground truth labeling, and storage for sensor data and trained classifiers. In the application, the user configures and chooses which sensors are available for sharing, selects neighbors with whom to share, and starts and stops classification. Figure 6.7 depicts activity feedback during runtime, indicating shared neighbors, expected activity duration, and time each neighbor has spent sleeping or awake. Figure 6.8 shows, in realtime, which sensors on which nodes are used in shared classification. We also provide a dialog to prompt the user to label ground truth before training a classifier.
With a web-based application, depicted in Figure 6.9, users can visualize individual and shared activity inferences. Each BSN user is able to see how his or her activities and locations intertwine with friends and colleagues. For example, in the figure, two users arrive separately on a university campus, conduct a meeting, and then leave together.

6.3 Neighbor Management

Neighbor Management allows a BSN to detect and address the dynamics of neighbor availability, initiating resource sharing only when it is beneficial. First, Proximity Detection detects potential sharing neighbors and ensures that all are performing the same activity. Next, through the use of a shared online calendar, Duration Prediction estimates the proximity duration of detected neighbors. Third, Cost-Benefit Analysis integrates a time and energy model with the predicted duration to determine if sharing with a detected neighbor will provide an energy benefit.

6.3.1 Proximity Detection and Duration Prediction

In our implementation, each BSN detects neighbors through overhearing neighbors’ on-body sensor data transmissions, however, detection can also be performed through phone-based Bluetooth discovery. Our neighbor detection scheme compensates for transmission delays or short periods of disconnection, such as when a neighbor briefly leaves a room and returns. We determine that a neighbor is in proximity if we overhear at least two packets from each neighboring mote within the last five aggregation intervals, or 50 seconds. If neighbors are detected, the local BSN exchanges the current activity with each neighbor to ensure all BSNs are performing the same activity. As mentioned previously, sharing neighbors must
be performing the same activity so that neighbors can duty cycle classifiers. While duty cycling, one active classifier makes a single classification decision for all neighbors, allowing inactive neighbors to sleep.

Proximity Detection is also responsible for determining when to stop sharing. If a local BSN is currently sharing with a neighbor that is no longer detected, we inform the Sharing-Aware Classification module to stop sharing and revert to individual classification. Similarly, if the local BSN is sharing as the active classifier and the currently classified activity changes, it informs all neighbors to stop sharing as they wake up, for the activity change may not hold for the other sleeping BSNs.

If at least one detected neighbor is performing the same activity as the local BSN, we then estimate the proximity duration of the detected neighbors to determine if the neighbors will be present long enough for sharing to benefit. Before runtime, each BSN user defines events in a shared online calendar outlining his or her expected activities. The calendar is updated during runtime based on classified activities. When neighbors are detected during runtime, the current time and locally classified activity is compared against the current calendar activity for each BSN. If all BSNs have a calendar entry for the currently classified activity, the expected duration is computed as the earliest finishing time of any current calendar entry.

We use shared calendars to initiate the bootstrapping process when no large scale proximity information is available. The calendar approach allows new users to deploy Remora without collecting any proximity information a priori. Note that the shared calendars are only a rough approximation of expected activities; they are not meant to be exhaustive or completely accurate and are used only as a guide in determining if sharing will be benefi-
cial. Similarly, previous work [85] investigates the use of shared calendars as sensors and concludes that with extra context information, such as classified activities, the calendar can be a powerful tool for monitoring human interaction. When an extensive history of interactions is available, we can use it to increase proximity prediction accuracy and provide a more holistic approach.

If at least one BSN does not have a calendar entry or is not performing the currently classified activity, the local BSN prompts the user via a dialog to ask if he or she expects all neighbors to be together long enough to share. Based on the cost model, in the sharing dialog, Remora will supply the user with the exact time length required for sharing to benefit. This manual sharing decision will bypass Cost-Benefit Analysis and, based on the yes/no decision of the user, will either directly initiate sharing or exclude the newly detected neighbor from shared classification.

6.3.2 Cost-Benefit Analysis

After a neighbor is detected and using the estimated neighbor proximity duration, we use a cost and benefit model to determine if sharing will result in energy savings. In our evaluation, most on-body motes ran without battery replacement during the two week experiment while phones had an approximate battery life of 10 hours for individual classification. Since phone battery life is the limiting factor in BSN lifetime, we focus on improving it through classifier duty cycling and Cost-Benefit Analysis. We define the costs in terms of time and energy to train a new classifier if a classifier has not already been trained using the current neighbors’ resources. We define the benefits in terms of energy saved during low power duty cycling among neighbors compared with always-on individual classification. In
Section 6.4.4, we empirically determine the cost model parameters and extend the model to our specific sharing design and BSN hardware.

We first define the phone energy required to collect new ground truth and train a new classifier while still performing individual classification, $E_{tr}$:

$$E_{tr} = [(T_{GT} + T_{tr})(P_{class} + P_{sensor})] + (T_{GT} \cdot P_{GT}) + (T_{tr} \cdot P_{tr})$$ \hspace{1cm} (6.1)

In Equation 6.1, $T_{GT} + T_{tr}$ refers to the total time needed to collect ground truth and train a new classifier. $P_{class}$ refers to the base power required to perform individual classification, while $P_{sensor}$ is the power consumed by sensors on the phone, including GPS and radio connectivity. Additionally, $P_{GT}$ and $P_{tr}$ refer to the additional power needed to collect ground truth and train a new classifier, respectively.

Next, we define the phone energy required to perform shared classification with neighboring resources, $E_{share}$:

$$E_{share} = [T_{prox} - (T_{GT} + T_{tr})] \cdot \left[ \frac{1}{b}(P_{class} + P_{sensor}) + \left(1 - \frac{1}{b}\right)P_{sleep}\right]$$ \hspace{1cm} (6.2)

In Equation 6.2, $T_{prox}$ is the predicted proximity duration, with $T_{prox} - (T_{GT} + T_{tr})$ representing the estimated time spent in shared classification after subtracting the time needed to collect ground truth and train a classifier $T_{GT} + T_{tr}$. Also, $b$ is the number of sharing BSNs, and $P_{sleep}$ is the power consumed by a BSN while it is in a low power sleep state. Note that each BSN spends an equal amount of time classifying; in this chapter, we ensure energy use fairness and leave a lifetime optimization scheme to future work.
Third, we define the phone energy required to classify as an individual BSN, $E_{ind}$:

$$E_{ind} = T_{prox} \cdot (P_{class} + P_{sensor})$$  \hspace{1cm} (6.3)$$

Equation 6.3 predicts the energy consumed by a BSN if it spends the expected proximity duration in individual classification instead of shared classification.

Lastly, using the above equations, we share when the energy to train a classifier and perform shared classification is less than performing individual classification for the expected proximity duration:

$$T_{prox} > (T_{tr} + T_{GT}) \text{ and } E_{tr} + E_{share} < E_{ind}$$  \hspace{1cm} (6.4)$$

In Equation 6.4, we also ensure that the predicted proximity duration is longer than the time needed to collect new ground truth and train a classifier. If a neighbor is detected and the above conditions hold, sharing is initiated by notifying Sharing-Aware Classification, which we describe next.

### 6.4 Sharing-Aware Classification

In this section, we first explain details on our classifier as well as what happens when a neighbor is detected and Cost-Benefit Analysis initiates sharing. Second, if sharing is initiated and a new classifier is needed, we then explain how neighboring BSNs train new classifiers by collaborating to determine a set of sensors to use for shared classification. Third, we explain how BSNs share classifiers and duty cycle them to significantly increase phone battery life. Lastly, through experimentation, we derive a cost model specific to our shared sensor selection approach and hardware.
We use an ensemble technique, Bootstrap Aggregating (Bagging) [15] for activity classification. Bagging is a lightweight approach appropriate for phones that makes classification decisions based on the majority vote of an ensemble of weak classifiers. In our Bagging classifiers, each weak classifier is a Naive Bayes classifier trained from the training data of a single sensor [67]. Other sharing approaches use more complex techniques, such as GMMs trained offline [89] or Boosting [76].

Bagging is exceptionally useful for addressing the dynamics of available neighbors: in addition to its quick training time and unlike many other classification methods, we can efficiently combine two existing Bagging classifiers into one large classifier, which we exploit during Collaborative Sensor Selection. Specifically, during Collaborative Sensor Selection, BSNs first train Bagging classifiers for individual sensor classifiers by training an ensemble of weak classifiers from the training data of a single sensor. Then, BSNs choose the best sensor classifiers and integrate them into a single composite classifier (classifier in previous sections) which is used to make runtime decisions for either an individual BSN or both local and neighbor BSNs.

**Runtime and Sharing Initialization.** At the start of runtime, each BSN either trains a new composite classifier for individual classification or loads a previously trained classifier from flash storage. Initial training is performed using Collaborative Sensor Selection but using local sensors only. During runtime, following detection of a neighbor, Cost-Benefit Analysis initiates sharing when it determines that sharing is beneficial. In a radio packet, a BSN announces an intent to share to its neighbors. Neighboring BSNs receive the packet, perform their own Cost-Benefit Analysis, and return a reply to indicate if they will participate. If at least two neighbors agree to share, either a previously trained classifier is reused
or more training data is collected and a new classifier is trained through Collaborative Sensor Selection.

**Classifier Reuse.** Composite classifiers are stored in flash memory for reuse. If a combination of neighbors meet, train classifiers, perform shared classification, and later meet again while performing the same activity, the previously trained classifiers are loaded from flash and used again. This saves significant sharing training and energy costs and allows sharing for short periods of time (5-10 min. in evaluation) with the same combination of neighbors and activities.

**Ground Truth and Sensor Classifier Training.** If neighboring BSNs do not have previously trained composite classifiers or the current activity is different than what these neighbors last performed, the neighbors collect new training data for the current activity. Training data is labeled with the current activity by the user and is collected for all local sensors and public neighbor sensors. When enough data is collected (5min in evaluation), each neighbor trains a sensor classifier for each available sensor and broadcasts its intent to start Collaborative Sensor Selection.

### 6.4.1 Sensor Selection Motivation

Before we explain our Collaborative Sensor Selection algorithm, we first provide intuition for its design using data from the motivation experiment. The challenge is to identify properties of both local and neighbor sensors such that, based on these properties, all neighbors can choose sensors to create composite classifiers that are accurate for all neighbors. Previous work [142] demonstrates that in order for ensemble classifiers, such as Bagging, to be trained successfully, two properties must hold: 1) the individual weak classifiers must
be accurate, and 2) weak classifiers must produce diverse classification results. We analyze these properties as they pertain to choosing sensor classifiers and adding them to a composite classifier. We conclude that choosing sensors based on individual accuracy (Figure 6.10) and decision correlation (Figure 6.11) will create an accurate composite classifier with a small number of capable sensors.

![Figure 6.10: Subject 1: Individual sensor accuracy.](image)

We first show in Figure 6.10 that we can discriminate best between activities by choosing sensors with the best individual accuracies. Using the motivation data, we train sensor classifiers for each sensor available to Subject 1, including publicly shared sensors from Subject 2. Each sensor classifier comprises 30 weak classifiers and we plot the average runtime accuracy of 10 classifier trainings on the y-axis. Each sensor is labeled by its on-body node ID and modality, where IDs starting with 1 are sensors from Subject 1 and IDs starting with 2 are sensors from Subject 2. Ranked by individual sensor accuracy, we can see that sensors on both Subject 1 and 2 exhibit the highest accuracy, indicating that sharing gives Subject 1 more accurate sensors from which to choose. Furthermore, we can see that the light, accelerometer, and microphone sensors perform the best, for they are able to distinguish between indoor and outdoor light levels, limb movements (walking), and
sounds, such as from a TV.

![Figure 6.11: Decision correlation and accuracy improvement.](image)

Next, in Figure 6.11, we illustrate how to find sensors with complimentary classification capability, locating a combination of sensors that is not only accurate but contains few sensing redundancies. In the figure, we show that when we add a sensor classifier to an existing composite classifier consisting of many other sensor classifiers, the greatest accuracy increase is observed when the sensor and composite classifier have uncorrelated classification decisions. The figure shows the decision correlation between a composite classifier generated from the data of 10 random sensors and a classifier generated from a single sensor not used by the randomly created composite classifier. We compute the accuracy change of combining the sensor classifier with the composite classifier. To compute decision correlation, each correct runtime decision is recorded as 1 and each incorrect decision is recorded as 0. From the figure, which contains 340 random composite classifiers, we can determine that by choosing sensors with decision correlation close to zero, we will ensure that each sensor we choose will produce a meaningful contribution towards an accurate composite classifier.
6.4.2 Collaborative Sensor Selection

Based on the motivation experiment, we provide a collaborative approach to training a composite classifier for shared classification. This approach is also used by a single BSN to train a composite classifier for individual classification when no neighbors are present. The main idea is for neighboring BSNs to iteratively add one sensor classifier at a time to their respective composite classifiers. At each iteration, all neighbors agree on a sensor classifier to add to their composite classifiers based on sensor classifier accuracy and decision correlation. A neighbor participates in sensor selection until it either maximizes training accuracy or all available sensors are added to its composite classifier. Using Algorithm 9, Collaborative Sensor Selection is explained in detail:

Each BSN \( i \) in the set of neighbors \( B \) first initializes a null composite classifier \( C_i \) (line 1 in Algorithm 9). Then, each BSN transmits to its neighbors the accuracies of each trained sensor classifier \( \text{acc}(s_j) \). Then, after all accuracy values are exchanged, each BSN ranks each sensor classifier \( s_j \in S \) by the following weight, \( w(C, s_j) \), in Equation 6.6.

\[
\begin{align*}
    w_i(C_i, s_j) &= \alpha \cdot \text{acc}_i(s_j) + (1 - \alpha) \left( 1 - r_{C_i, s_j} \right) \\
    w(C, s_j) &= \frac{1}{B} \sum_{i=1}^{B} w_i(C_i, s_j)
\end{align*}
\]

In Equation 6.6, each sensor classifier for BSN \( i \) and sensor \( s_j \) is weighted by its accuracy, decision correlation \( r \) with the current composite classifier \( C_i \), and the number of BSNs \( B \). \( \alpha \) provides a weight to emphasize either accuracy or decision correlation when weighting (we use \( \alpha = 0.5 \)). At first, when the composite classifier is null, each sensor classifier is weighted only by accuracy. Also, if the classifier is for a private sensor, no neighbors have
Algorithm 9 Collaborative Sensor Selection

Input  Sensor classifiers for local sensors and public neighbors $S$, Sensor classifiers for private sensors $P$

Output  Composite classifier for local BSN $C_i$

// BSN initiates sharing
1: function START($S$)
2:   $C_i = \emptyset$
3:   Send to neighbors: $\forall s_j \in S, \text{acc}_i(s_j)$
4: end function

// Receive sensor classifier accuracy or correlation values from all neighbors; choose a sensor classifier based on weights and neighbor composite ensembles $C$
5: event CHOOSESENSOR($S, C$)
6:   $\forall s_j \in S$ compute weight $w_j$ with Equation 6.6
    // First, add private sensors with highest weight
7:   repeat
8:      Get $W$, sensor classifiers with highest weight
9:      for all $w_i \in W \cap P$ do
10:         ADDSENSOR($w_i, S, C_i$)
11:      $W = W \setminus w_i$
12:   end for
13: until $|W| > 0$
    // Next, choose public sensor
    // Only one sensor $w_0 \in W$ has highest weight, add it
14: if $|W| = 1$ then
15:   ADDSENSOR($w_0, S, C_i$)
    // Several sensors with same weight; Local BSN wins tiebreaker
16: else if local BSN ID is the lowest of all neighbors then
17:      Choose random classifier $w_r \in W$
18:      ADDSENSOR($w_r, S, C_i$)
19:      Notify neighbors of choice $w_r$
    // Several sensors with same weight; Local BSN loses tiebreaker
20: else
21:      Wait for tiebreaker BSN to choose random classifier $w_r \in W$
22:      ADDSENSOR($w_r, S, C_i$)
23: end if
24: if $\text{acc}_i(C_i) < 1$ and $|S| > 0$ then
25:      Send to neighbors: $\forall s_i \in S, \text{acc}_i(s_j)$
26: end if
27: end event
    // Add sensor $s_j$ to composite classifier $C_i$ if $s_j$ improves accuracy
28: function ADDSENSOR($s_j, S, C_i$)
29:     if $\text{acc}_i(C_i \cup s_j) > \text{acc}_i(C_i)$ then
30:        $C_i = C_i \cup s_i$
31:     end if
32:     $S = S \setminus s_i$
33: end function
accuracy or decision correlation information for the classifier, so the weight is computed using Equation 6.5 only.

After computing weights (line 5 in Algorithm 9), each BSN then chooses the sensor classifier with the highest weight. Since each BSN computes the same weight values independently, each BSN will choose the same sensor classifier. However, if there are multiple sensor classifiers with the same weight, the BSN with the lowest BSN ID value chooses a sensor and broadcasts its choice to neighboring BSNs (lines 14-17). If a private sensor (sensor a user does not share with neighbors) classifier has the highest weight, it is chosen along with one other public sensor classifier to ensure all BSNs choose the same classifier (lines 6-11). Once a sensor classifier is chosen, it is only added to the composite classifier if it increases the composite classifier accuracy.

After a sensor classifier is chosen, a BSN stops sensor selection if there are no more remaining sensors to choose from or the BSN has achieved perfect training accuracy. While adding more sensor classifiers to a composite classifier with perfect training accuracy may improve runtime accuracy [15], we focus on reducing training costs and stop when we achieve perfect training accuracy. Remaining BSNs then compute decision correlation $r$ between the ensemble classifier and each remaining sensor classifiers and broadcast the correlation values. Another sensor classifier is then chosen in the manner above and the process repeats.

### 6.4.3 Classifier Sharing and Duty Cycling

After sharing is initiated by Cost-Benefit Analysis and a classifier is trained or reused, all neighbors collaborate to define a duty cycle order where at any given time period, only one phone is active and classifying activities. Neighbors exchange a random integer
concatenated by a BSN ID integer, with the duty cycle order following the ascending order of the generated values. While we can optimize duty cycle periods for each phone to maximize lifetime, we leave that to future work and use a round robin duty cycling scheme to ensure fairness in energy consumption. We choose a duty cycle of appropriate length (5 min. in evaluation) so that sleeping neighbors are able to timely wake up and detect changes in available neighbors, especially if the active classifier departs.

After a new classifier is trained, it is saved to flash storage to facilitate both reuse and duty cycling, for when a phone goes to sleep, memory may be purged. Upon waking up as the classifying neighbor, a BSN loads the classifiers back into memory including the one for the current neighbor combination. Saving and loading is nearly instantaneous, requiring little overhead. Upon waking up, if at least one sharing neighbor has departed, the BSN reverts to individual classification and notifies all remaining sharing neighbors.

6.4.4 Empirical Sharing Cost Model

Using Collaborative Sensor Selection, we empirically define a sharing cost model specific to our system which extends the general model defined in Section 6.3.2. Using an HTC Hero smartphone and four on-body sensors as described in Section 6.1.2.1, we perform time and power benchmarks of our Remora implementation. We use the benchmarks to define the training time, training power, and minimum proximity duration needed for sharing to provide an energy benefit.

We measure power consumption by connecting an HTC Hero smartphone-based BSN running Remora to a Monsoon Technologies Power Meter, demonstrating that we achieve massive phone energy savings by duty cycling classifiers. In Figure 6.12, we create a power
profile of each system behavior with WiFi and GPS disabled. The power meter setup is depicted in Figure 6.13. At time 0, we show the idle power consumption of the phone with CPU active and start individual classification at 150s. Touching the screen to start classification briefly consumes about 700mW. Classification consumes an additional 150mW of power until a neighbor is encountered at 230s and sharing starts. The user labels ground truth three times (screen usage spikes) over five minutes until enough new training data is collected to train a shared composite classifier at 570s. Training while performing individual classification lasts more than 5 min. and consumes about 200mW more than classification alone, and upon completion of training, the phone immediately goes to sleep with the neighbor actively classifying for both BSNs. A sleeping phone consumes fewer than 10mW of power, which is much less than the nearly 500mW required for classification.

<table>
<thead>
<tr>
<th>Base Power</th>
<th>Additional Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classify</td>
<td>Sleep</td>
</tr>
<tr>
<td>486.43</td>
<td>5.25</td>
</tr>
<tr>
<td>Ground Truth</td>
<td>Train</td>
</tr>
<tr>
<td>+47.62</td>
<td>+88.51</td>
</tr>
</tbody>
</table>

Table 6.2: Remora Power Consumption (mW).

The average power consumption for each state is provided in Table 6.2: base power
for sleep or classification (consumption is the same for individual and shared classification), and additional power required for collecting ground truth (screen use), training a composite classifier through Collaborative Sensor Selection, and GPS and WiFi use.

![Graph showing training time vs. number of sensors](image)

**Figure 6.14:** Phone training time: sensor classifiers and Collaborative Sensor Selection

On the phone, we also measure the training time required to train new sensor classifiers and perform Collaborative Sensor Selection. We vary the number of sensors given to Collaborative Sensor Selection and plot the results in Figure 6.14. Our training algorithm has polynomial time complexity: with respect to the number of sensors $n$, training sensor classifiers is $O(n)$. Collaborative Sensor Selection is $O(n^2)$: each time a sensor classifier is added to the composite classifier, decision correlation is computed for each unchosen sensor classifier. From the figure and training time data points, we perform polynomial curve fitting to determine the training time $T_{tr}(n)$ in Equation 6.7 for use in our cost model, where $n$ is the number of sensors.

$$T_{tr}(n) = 167.7n^2 + 2053n + 7118$$ (6.7)

The polynomial time complexity indicates that training times are significantly faster when providing fewer sensors as training input. When training an individual composite
classifier, we give all local sensors as input (20 on-body and 2 phone), which requires about two minutes of training. However, if a BSN uses all local and public neighbor sensors when training a shared composite classifier (42 for 2 BSNs), training can take more than 7 minutes. With more neighbors, training can take even longer. Instead, when neighbors are present, we ensure that only sensors chosen by each BSN for individual classification are given as input for shared classification. This ensures that shared training overhead is reduced. Furthermore, Figure 6.14 and Equation 6.7 illustrate that sharing with a fewer number of neighbors will provide a greater energy benefit and also allow sharing over shorter durations due to the lower training costs.

![Figure 6.15: Proximity duration needed for sharing to benefit.](image)

In Figure 6.15, we provide more evidence that sharing with a small number of neighbors is most beneficial. Using Equation 6.7, our power consumption results, and our cost model from Section 6.3.2, we compute the minimum proximity duration needed for sharing to provide an energy benefit. In our evaluation, each BSN uses 10 sensors on average for individual classification, which is the input size when building a shared classifier. This indicates that the minimum proximity duration is under 20 minutes for up to 5 neighbors,
which is reasonable under most circumstances. However, with 10 neighbors, training more than doubles to over 40 minutes, which is impractical for many dynamic scenarios.

### 6.5 Evaluation

We evaluate our Remora shared activity classification system using the configuration and data collection methods in Section 6.1.2.1. We combine the motivation experimental data with two weeks of new data. In the evaluation, six subjects, including the first two subjects from the motivation, perform a combination of the following activities: riding a bus, riding in or driving a car (driving), meeting, reading, running, watching TV, walking, and working at a desk. The subjects we use meet our design target of sharing with neighbors with strong interpersonal ties: all are graduate students or family members who spend a significant amount of time together. Each subject has an initial training set of 30 observations per activity (5 min.) and trains an individual classifier at the start of runtime, with each sensor classifier trained using 30 weak classifiers. We evaluate three scenarios using the same data and activity ground truth: individual classification only, sensor sharing only, and sensor and classifier sharing. In Section 6.5.1, we first look at sharing opportunities. Then, we demonstrate sharing accuracy improvements in Section 6.5.2. In Section 6.5.3, we highlight the benefits of sharing as well as significantly improved battery life in Section 6.5.4. Lastly, in Section 6.5.5, we highlight Remora CPU and memory usage.

#### 6.5.1 Proximity and Sharing

In this section, we show that subjects are in proximity to each other for a significant portion of time, giving ample sharing opportunity. We also show that Remora uses 96% of the total
proximity duration to share sensors and classifiers.

<table>
<thead>
<tr>
<th>Subj.</th>
<th>Prox. Duration (%Total)</th>
<th>Reuse (% Prox. Duration)</th>
<th>No Reuse (% Prox. Duration)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22h36m (39%)</td>
<td>21h30m (95%)</td>
<td>17h42m (78%)</td>
</tr>
<tr>
<td>2</td>
<td>04h06m (82%)</td>
<td>03h54m (96%)</td>
<td>03h34m (87%)</td>
</tr>
<tr>
<td>3</td>
<td>10h30m (65%)</td>
<td>10h06m (96%)</td>
<td>08h39m (82%)</td>
</tr>
<tr>
<td>4</td>
<td>09h18m (64%)</td>
<td>08h48m (95%)</td>
<td>07h15m (78%)</td>
</tr>
<tr>
<td>5</td>
<td>06h06m (63%)</td>
<td>05h54m (96%)</td>
<td>04h39m (76%)</td>
</tr>
<tr>
<td>6</td>
<td>07h06m (68%)</td>
<td>06h42m (95%)</td>
<td>04h57m (70%)</td>
</tr>
</tbody>
</table>

Table 6.3: Total Proximity and Sharing Duration.

First, in Table 6.3, we show the total time at least one neighbor is in proximity for each subject (Prox. Duration). While the average percentage of time sharing is greater than the MIT dataset [31], the subjects we evaluate are a close knit group of students and family. Furthermore, since the focus of the evaluation is on sharing performance, most of the subjects do not wear the BSN for all individual activities. However, Subject 1 performs a large number of individual activities and is more representative of the neighbor proximity duration that would be expected from such a group.

Next, Table 6.3 demonstrates when we perform sharing with classifier reuse (Reuse) that an average of 96% of the total proximity time is utilized. The remaining 4% difference is due to Cost-Benefit Analysis rejection due to short proximity durations or different simultaneous activities. The time difference also includes sharing overhead: time to collect ground truth and perform sensor selection. With classifiers reused for multiple encounters with the same neighbor combination, Remora can quickly adapt to share with available neighbors. Also, classifier reuse accounts for 90% of sharing encounters among all BSNs. Without classifier reuse (No Reuse), however, sharing overhead is much higher, with an average of 78% of the total proximity time utilized.
6.5.2 Accuracy Improvement

![Figure 6.16: Runtime accuracy for each subject.]

![Figure 6.17: Subject 1 accuracy.]

![Figure 6.18: Subject 4 accuracy.]

![Figure 6.19: Subject 6 accuracy.]

In Figure 6.16, we highlight overall accuracy performance for each BSN for individual classification, sharing sensors, and sharing both sensors and classifiers, analyzing only the periods where sharing is possible to make a fair comparison. From the figure, all subjects except for Subject 2 receive an accuracy benefit from sharing sensors and classifiers, with Subjects 1, 4, and 6, receiving the greatest accuracy gains of over 20% points, or nearly 30% over individual classification. The figure also demonstrates that duty cycling classifiers among neighboring BSNs has no impact on accuracy. The 5 min. duty cycle period is short enough for each BSN to capture changes in its own activities as well as neighbor departures.
and stop sharing if such a change is detected.

From Figure 6.16, Subject 2 has the highest accuracy of all subjects, which is because Subject 2 does not perform as many activities as the other subjects. Conversely, Subject 1 performs a multitude of activities and has the lowest individual accuracy. In Figure 6.17, we can see that Subject 1 exhibits confusion between the meeting and working activities in addition to the reading and watching TV activities presented in Section 6.1.2.2. The additional sensors provided by neighbors are able to overcome these challenging activities. Similar confusion between meeting and working can also be witnessed in Figure 6.18 and 6.19 for Subjects 4 and 6, respectively, where total accuracy is also significantly improved by sharing. Lastly, we note that Collaborative Sensor Selection ensures that accuracy is only improved by sharing; high individual accuracy is maintained during sharing while low individual accuracy is improved.

![Figure 6.20: Accuracy improvement over individual classification for each BSN, activity, and neighbor combination. For example, the first column is the accuracy increase for BSN 1 when sharing with BSN 2. Black indicates no data.](image)

In Figure 6.20, we show the accuracy improvement of sharing sensors and classifiers over individual classification. The figure compares BSN performance for each activity and neighbor combination. The figure demonstrates that the neighbors with the worst individual
accuracies gain the most accuracy benefit by sharing. For example, Subject 1 gains over 30% points accuracy in the meeting activity when sharing with Subjects 5 and 6, which is also reflected in the accuracy improvement for all shared meeting activities in Figure 6.17. Similar improvements are also witnessed for Subjects 4, 5, and 6. While Subjects 2 and 3 do not have as high accuracy gains by sharing, these two subjects have high individual accuracy, and thus the marginal improvement is lower.

6.5.3 Sharing Costs and Benefits

We evaluate Neighbor Management and Cost-Benefit Analysis to show that we can accurately determine when and how much sharing will benefit. In Figure 6.21, we plot the sharing decision accuracy when using the shared calendar or manually asking the user for proximity duration when the calendar has no result. Accuracy is computed as the number of true positive sharing decisions (sharing initiated and lasted long enough to achieve an energy benefit) plus true negative sharing decisions (sharing not initiated and proximity duration was too short for sharing to benefit) compared with the total number of sharing decisions. Both the calendar and manual prediction have similar performance and achieved
over 90% accuracy for all subjects. We also show that the calendar is used to make between 50% and 70% of all sharing decisions, which significantly reduces user invasiveness since Remora does not have to ask the user to estimate proximity duration.

Next, we illustrate that for every subject, the benefits of sharing outweigh the costs by two orders of magnitude. First, from our cost model, in Figure 6.22, we compute the ratio between the energy savings gained through duty cycling classifiers and the additional energy costs required to collect ground truth and train shared classifiers. The average net energy savings for the shared periods is about 400% for all subjects. During the experiment, each subject was in proximity with no more than two neighbors simultaneously, however, we see that the bulk of the energy savings comes from sharing with one neighbor. The marginal benefit of sharing with an additional neighbor is negative with the exception of Subject 4. This demonstrates that sharing with a small number of BSNs achieves high accuracy with low cost; sharing with a large number of BSNs is impractical and will be rejected by our Cost-Benefit analysis.

6.5.4 Energy Savings

![Image](image_url)

**Figure 6.23:** Battery Life.
We now demonstrate that by sharing and duty cycling classifiers, we can increase phone battery life up to 65%. We also show that we can save mote energy while sharing sensors to reduce the number of sensors needed by nearly 50%. To compute battery life for each BSN, we determine as a percentage of the total running time: time spent during active classification and sleep, phone sensor use, training time and ground truth labeling. Combined with power consumption in Table 6.2 and a 1500mAh battery per phone, we present results in Figure 6.23. For each BSN, individual classification yields about 10 hours of battery life. However, with duty cycling through classifier sharing, battery life can be extended from 13 hours for Subject 1 to almost 17 hours for Subject 4. This represents an increase ranging from 25% to over 65%.

![Graph showing energy and accuracy profile](image)

**Figure 6.24:** Sharing timeline: energy and accuracy profile.

To further highlight classifier sharing as well as the ability of Remora to adapt classification to available neighbors, Figure 6.24 presents a timeline of a shared classification period between Subject 1 and 2. In the figure, we present power usage as well as mark correct and incorrect activity classifications for each classification decision. Subject 1 and 2 perform individual activities until 7 minutes, where Subject 2 enters a building after being outside.
and meets Subject 1 (note that the GPS is active and consumes more sensor energy). After Subject 1 and 2 meet, Proximity Detection and Duration Prediction estimates the length of the proximity period while Cost-Benefit Analysis quickly determines that sharing will provide an energy benefit, initiating ground truth labeling and classifier training. During the individual periods, Subject 1 makes many misclassifications but after ground truth is logged and a new classifier trained, Subject 1 exhibits high accuracy with no misclassifications. After training is complete at 14 minutes, both BSNs trade off as the active classifier, alternately going to sleep until Subject 1 leaves and goes outside, returning to individual classification with several misclassifications.

![Graph showing percentage of available sensors chosen](image)

**Figure 6.25:** Percentage of available sensors chosen.

While accuracy is increased over individual classification, sharing sensors can also reduce the total number of sensors used by all neighbors. In Figure 6.25, we plot the average percentage of available sensors chosen during Collaborative Sensor Selection. The figure shows that between 10% and 20% points fewer sensors are used while sharing sensors or classifiers compared with individual classification. This is because Collaborative Sensor Selection is able to identify and use only the sensors that provide the most accuracy benefit. When neighbors are present, there are more sensors to choose from and more sensors that
provide a large contribution towards providing high classification accuracy.

6.5.5 CPU and Memory Usage

<table>
<thead>
<tr>
<th>Mode</th>
<th>CPU</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classify</td>
<td>23%</td>
<td>8.1MB</td>
</tr>
<tr>
<td>Classify + Ground Truth</td>
<td>23%</td>
<td>8.1MB</td>
</tr>
<tr>
<td>Classify + Train</td>
<td>96%</td>
<td>9.2MB</td>
</tr>
<tr>
<td>Sleep</td>
<td>0%</td>
<td>8.1MB</td>
</tr>
</tbody>
</table>

Table 6.4: Remora CPU and memory benchmarks.

In Table 6.4, we illustrate CPU and memory usage as determined by the Android SDK toolkit. Since classifiers for the same neighbor combination are reused once trained, most of the time Remora incurs about 23% CPU overhead during classification. While training, Remora maximizes CPU availability, however, this lasts no more than 10-15 minutes depending on the number of sensors and BSNs available. Furthermore, because Remora training and classification is run as a background process, the CPU scheduler gives priority to other applications running in the foreground so that the phone remains responsive. Lastly, memory usage remains relatively constant during classification as well as during training, ranging from about 8-9MB.

6.6 Conclusion and Future Work

In this chapter, we propose Remora, a smartphone-based body sensor network system for activity classification which exploits physical proximity of neighboring BSNs to provide increased accuracy and energy savings. First, through a time and energy cost-benefit analysis, we determine when sharing provides an energy benefit. Second, our Collaborative Sensor
Selection approach efficiently chooses a small number of sensors that provides high accuracy for all shared BSNs. Third, classifiers among sharing neighbors are duty cycled to provide a significant boost in phone battery life. Our multi-week evaluation demonstrates an accuracy improvement of up to 30% and battery life improvement of over 65%. In future work, we propose to maximize phone battery life with an optimal duty cycling scheme while also considering energy use of other applications. We also intend to extend sensor data and classifier sharing to the cloud to further improve accuracy, extend sharing support to social networking applications, and investigate duty cycling classifiers when neighbors perform different activities.
Chapter 7

Sidewinder

In many wireless sensor deployments, only a small number of sink nodes may be directly connected to an end user receiving confident sensing decisions. Furthermore, the deployment configuration may be the most effective for confident sensing if nodes forward sensor data to an aggregator or cluster head node through multiple hops. Many applications for confident sensing use mobile nodes: deployments of vehicles [33], animals [126], and unmanned aerial vehicles [3] are common in providing solutions for traffic congestion control, animal tracking, and military surveillance. Existing wireless sensor routing protocols either assume nodes are static [139] [129] [63] [70] [108] [48] [37] or provide a delay tolerant solution [55] [33] [119] [32]: neither is desired for performance critical applications with mobile deployments. Furthermore, solutions for general mobile ad hoc wireless networks [97] [38] [59] [72] [10] do not address the volatile topology changes present in mobile wireless sensor networks.

Consequently, we propose Sidewinder, a predictive data forwarding protocol for mobile wireless sensor networks. Like a heat-seeking missile, data packets are guided towards a sink node with increasing accuracy as packets approach the sink. Different from conventional
sensor network routing protocols, Sidewinder continuously predicts the current sink location based on distributed knowledge of sink mobility among nodes in a multi-hop routing process. Moreover, the continuous sink estimation is scaled and adjusted to perform with resource-constrained wireless sensors. We have published our research results in [64], and in this chapter, we address the following challenges:

- Performance analysis of traditional in-situ data collection with excessive topology changes due to sensor mobility. We perform a quantitative evaluation of traditional mobile ad hoc and wireless sensor routing protocols and determine that increased network dynamics cause the performance of these protocols to degrade significantly.

- Handling excessive topology changes in wireless sensor networks. Using Sequential Monte Carlo (SMC) theory, the Sidewinder solution we propose aggregates distributed knowledge of a mobile sink location among intermediate nodes to ensure a correct and successful forwarding path.

- A forwarding solution appropriate for low power wireless sensor networks. The SMC prediction approach in Sidewinder is scaled to reduce computation and bandwidth overhead.

This chapter is organized as follows: Section 7.1 motivates this work with simulation. Section 7.2 gives an overview of the Sidewinder design. In Section 7.3, we discuss the Sequential Monte Carlo Prediction-based data forwarding strategy. The performance evaluation is presented in Section 7.4. In Section 7.5, we present conclusions.
7.1 Motivation

The results of the following quantitative studies illustrate new challenges that cannot be handled by traditional routing methods and motivate the design of our new routing protocol. In Section 7.1.1, we show the impact of radio ranges on topology changes when nodes are mobile, concluding that traditional mobile ad hoc routing protocols do not work well for wireless sensors. In Section 7.1.2, we show the detrimental performance impact of sink mobility on geographic forwarding-based protocols like GF [63] and its mobility-oriented enhancements [114].

7.1.1 Impact of Radio Ranges on Topology Changes

Taking AODV [97] as an example, we study how a routing protocol designed for general mobile wireless networks performs in a mobile wireless sensor network context. We implement AODV with nesC [41] in TOSSIM [98], and evaluate it with 500 nodes. A source-sink pair is chosen from the 500 nodes, and a multi-hop (~6 hops) path is established between them using AODV. We measure the path lifetime, which is defined as the time from which a path is established until it is broken, and study it under different node mobilities and radio ranges. For node mobility, we use the Random Waypoint [16] model without pause time [135]. We increase the radio range from 25m to 250m with 25m increments. To eliminate any effect of changing node density on end-to-end routing performance, we keep the node density at approximately 20 one-hop neighbors per node. To help focus on performance study of mobility, we do not simulate node failures or lossy links [139] [140] [128] so that their impact on routing performance can be separated.
Two interesting observations are identified in Figure 7.1. First, the multi-hop end-to-end link has a much longer lifetime when the radio range is 150 ~ 250m than when the radio range is 10 ~ 40m. For example, the observed link lifetime goes up to 109s when the radio range is 250m, but it reduces to 18s when the radio range is 25m. This 84% link lifetime reduction indicates that even though AODV works well in general mobile wireless networks, it leads to very poor performance when it is applied to mobile wireless sensor networks. Second, when the node mobility is high, AODV leads to poor performance in both general mobile wireless and sensor networks. For instance, when the node mobility increases to 5+m/s, the path lifetime reduces to ≤ 11 seconds in general mobile wireless networks and ≤ 3 seconds in sensor networks. Routes with such a short lifetime are useless in practical systems, therefore, routing protocols that are originally designed for general mobile wireless networks [97] [38] [59] [72] can not be directly applied in mobile wireless sensor networks. Thus, a new protocol is required for wireless sensor networks that can handle high node mobility and accurately guide data from source to sink.
7.1.2 Impact of Sink Mobility on Geographic Forwarding

Geographic forwarding-based protocols [63] [70] [108] [48] [37] have been widely used in static wireless sensor networks, because they only maintain local information to achieve end-to-end routing. However, a common assumption of these geographic forwarding-based protocols is that all intermediate nodes in a routing path know the exact sink location and use it for multi-hop routing. This assumption is reasonable when the sink is static, but leads to poor performance when the sink is mobile. In this experiment, we use the same TOSSIM configuration as presented in the previous experiment to evaluate one of the most representative geographic forwarding-based protocols, GF [63], in mobile environments. GF is implemented with the Destination Location Prediction (DLP) mobility enhancement in [114]: when a forwarding node is a one-hop neighbor of the sink, the forwarding node selects the sink as the next hop. DLP is designed to eliminate local maximums, where a forwarding node is a one hop sink neighbor, but the forwarding node neighbor table indicates that it is the node closest to the sink location.

![Graph](image)

**Figure 7.2**: Performance Impact of Sink Mobility on Geographic Forwarding

As shown in Figure 7.2, when all nodes are static, the end-to-end packet delivery ratio is 100%. However, when nodes are mobile, the end-to-end packet delivery ratio drops sharply.
For example, when the maximum node speed is 1m/s, which means a 0.5m/s average speed, the GF packet delivery ratio goes down to 50%. When the maximum node speeds increases further to 5m/s, 10m/s and 20m/s, the end-to-end packet delivery ratio reduces to ≤ 20%.

There are two reasons why GF performs so poorly when nodes are mobile. First, the geographic forwarding direction is not continuously corrected based on distributed sink mobility knowledge among nodes in the routing path, so a source may fail to find a route to a mobile sink. Second, it is too costly to keep updating a neighbor table to deal with mobility, and an inadequate update usually leads to failure of data forwarding. Neighbor Location Prediction, presented in [114], predicts neighbor locations based on movements and attempts to reduce the beaconing frequency; zone-based forwarding described in [14] presents a more efficient alternative without the use of a neighbor table. While zone-based forwarding is integrated into our Sidewinder design, our performance evaluation in Section 7.4 shows that by itself, it is still lacking.

We have shown that traditional ad hoc routing solutions suffer with frequent topology changes in mobile environments. We also illustrate that traditional wireless sensor routing solutions face routing difficulties in mobile environments due to reliance on local decisions and costly neighbor tables. To address these concerns, we propose the use of Sequential Monte Carlo (SMC) estimation to increase the prediction accuracy of the sink location over multiple hops. We are aware that SMC estimation has been used to improve tracking/localization [54] [42] and sensor fusion [44] performance. These previous studies also point out that SMC is a better choice than other prediction techniques, for it is more accurate than Kalman filters and incurs less overhead than Markov models. We integrate this SMC estimation technique into Sidewinder in a low-overhead manner to account for the
computational and bandwidth restrictions present in wireless sensor networks.

### 7.2 Sidewinder Overview

Our Sidewinder protocol lies between the transport and MAC layers, as shown in Figure 7.3. It gets data from the transport layer and forwards it to sink nodes. It also gets global time and individual node location information from the time synchronization and localization protocols, which have been extensively studied in wireless sensor networks. Any existing MAC protocols, like B-MAC [99], can be used below Sidewinder. The predictive multi-hop forwarding functionality is achieved through combined efforts of four modules: Sequential Monte Carlo (SMC) Prediction, Limited Flooding, Mobility Monitor and Adaptive Update.

![Sidewinder Architecture](image)

**Figure 7.3: Sidewinder Architecture**

The SMC Prediction module utilizes partial knowledge of sink location from all intermediate nodes in the routing path to make a combined sink location estimate, and hence makes the informed local data forwarding decision dynamically. This holistic prediction of sink location based on distributed knowledge among intermediate nodes facilitates in-situ data forwarding to a mobile sink. This module enables Sidewinder to achieve high packet delivery ratio and low time delay in mobile sensor network environments.
While the SMC Prediction module enables Sidewinder to forward data close to a mobile sink, the Limited Flooding module ensures that data finally reaches the sink. When a forwarding node is a one-hop neighbor of the sink node, data packets received are passed to the Limited Flooding module, which broadcasts them to all the node’s two-hop neighbors, ensuring the data reaches the sink even if the sink has moved out of range of the forwarder.

Since group mobility is a common characteristic of buoyant sensor networks or mobile sensor networks such as ZebraNet [82] it is important to measure individual nodes’ mobility as well as group mobility. Mobility can be measured based on nodes’ locations, which are obtained from localization protocols. The Mobility Monitor module serves this purpose, providing the measured ground truth for SMC Prediction. The Adaptive Update module provides a link between the Mobility Monitor and SMC Prediction, disseminating sink location and mobility information to the network. The Adaptive Update module updates the network in a time and space adaptive manner to save energy and reduce communication overhead.

We now explain in detail SMC Prediction, Mobility Monitor, and Adaptive Update. The Limited Flooding module is comparatively simple and hence not elaborated.

### 7.3 Detailed Sidewinder Design

In this section, we first discuss the general idea of how the Sequential Monte Carlo theory can be integrated into multi-hop data forwarding. Then, we discuss how this design can be trimmed to fit into sensor network environments, where bandwidth and energy are limited. Finally, we present details of each phase in SMC Prediction-based data forwarding along
with adaptive sink update.

7.3.1 SMC Prediction Concept

During multi-hop data forwarding toward a mobile sink, the goal of the SMC Prediction module is to determine the current sink location using possible locations estimated by previous hops as well as the current hop. It consists of four phases: initialization, prediction, filtering, and resampling.

In initialization, $N$ possible sink locations are generated by a source node based on the last heard sink location, group and random velocities. These $N$ locations form a possible location distribution of the mobile sink. In the prediction phase, a forwarding node uses both the $N$ sink locations generated by the previous node as well as its own knowledge of sink location, group and random velocities to predict current sink locations. The filtering phase allows a forwarding node to use both the previous node's and its own sink location predictions to eliminate impossible sink locations. Finally, in the resampling phase, a forwarding node uses its own sink location information to generate new possible sink locations to replace those eliminated in the filtering phase. In this way, a data packet can be forwarded towards a continuously corrected, predicted sink location, using the knowledge distributed on the nodes in the path.

Several papers [14] [49] [77] [144] show that to deal with frequent topology changes, it is better to let all nodes that overhear the forwarded data to compete for the next hop forwarding, rather than assigning a specific node among a frequently updated neighbor table to forward the data. We incorporate this zone-based forwarding wisdom in Sidewinder: if a node hears a data forwarding packet and also finds itself currently within the 60° forwarding
zone facing the sink, it competes for the next hop forwarding; Otherwise, it ignores the data packet. A $60^\circ$ forwarding area has been shown in [49] to reduce redundant forwarding, so we integrate this forwarding area size into Sidewinder. When a node competes to forward to the next hop, it sets a backoff timer with respect to the forward progress to the estimated sink area center. A competitor within this $60^\circ$ zone overhears this data forwarding and cancels its backoff timer.

7.3.2 SMC adapted to Wireless Sensor Networks

Since communication is far more expensive than computation in sensor networks, it is necessary to minimize the communication overhead of integrating SMC Prediction into multi-hop routing. The main overhead is to send the $N$ estimated sink locations to the next hop. If 4 bytes are used to represent a node location $(x,y)$, $4N$ bytes are needed to represent $N$ locations. Assuming $N = 8$ for achieving reasonable prediction accuracy, 32 bytes are needed for each data forwarding decision. Considering that a typical sensor network packet size is $\sim 40$ bytes, this is too much overhead, and hence alternative representations are needed.

One alternative is to consider a Convex Hull algorithm like Quickhull [9]. Instead of using all $N$ locations, QuickHull uses $Q$ edges to represent the potential sink area. However, Quickhull is not able to represent the real location distribution of the sink within the potential Convex Hull area, and hence reduces the prediction accuracy.

Another alternative is to consider clustering techniques, such as QT [50], that can group $N$ predicted sink locations into $Q$ clusters. To represent each cluster, 4 bytes are needed for the center location, 1 byte is needed for the radius, and 1 byte is needed for the number
of locations in this cluster. So representing $Q$ clusters needs $6Q$ bytes. If $Q = 3 \sim 4$, then 18~24 bytes are needed to forward a data packet in each hop, which is also too much overhead. So, a conventional clustering technique is also not a good choice.

Therefore, we propose a sector method that uses a one-dimension, rather than the conventional two-dimension, clustering technique to reduce communication and energy overhead while preserving the sink location distribution in the predicted area. Although reducing clustering from two dimensions to one loses some information, it does not produce a noticeable performance impact. This is because a multi-hop data forwarding decision only needs to know the orientation and distribution of the predicted sink locations from the forwarding node. The distances from the forwarding node to these possible locations are unnecessary. Therefore, in our sector method, a $60^\circ$ forwarding area focused on the estimated sink area center is used and is also split into $Q$ sectors, with the number of possible sink locations inside each sector calculated to represent the sink location distribution in the forwarding direction. It is effective to just transmit the number of possible sink locations in each sector, rather than all possible sink locations, so the total cost is reduced to just $Q$ bytes.

7.3.3 Mobility Monitor

To understand the SMC forwarding approach and its relationship with node mobility, we first describe the calculation of group and random velocities. We determine the general movement of network as a whole as well (group velocity) as individual nodes' random
deviations from that group movement (random velocity). Since there exists an extensive study of retrieving accurate sensor locations through GPS or localization algorithms, in Sidewinder, we assume that each node can get its own location from the localization module as denoted in Figure 7.3. Based on location history, each node can compute its own velocity. A sink node can also obtain its neighbors’ velocities through local beaconing, and combine them with its own velocity $\vec{v}_i$ to compute its group and random velocities. As shown in Figure 7.4, when sink S obtains its own velocity $\vec{v}_i$ and its neighbors’ velocities $\vec{v}_a$, $\vec{v}_b$, $\vec{v}_c$, $\vec{v}_d$, and $\vec{v}_e$, its group velocity can be computed as the averaged velocity: $\vec{V}_G = (\vec{v}_a + \vec{v}_b + \vec{v}_c + \vec{v}_d + \vec{v}_e + \vec{v}_i)/6$, and the its random velocity $\vec{V}_R$ can be computed as the difference between absolute and group velocities: $\vec{V}_R = \vec{v}_i - \vec{V}_G$. With this concept in mind, we now explain our SMC approach.

7.3.4 SMC Initialization

In initialization, a source node predicts sink locations. It also forwards the predicted locations together with application data towards a mobile sink. The source node’s neighbors overhear the forwarded data, and those currently lying in the specified 60° forwarding zone compete for the next hop forwarding.

As shown in Figure 7.5, a source node (A) computes an estimated sink area, based on the stored sink location (S), group velocity ($\vec{V}_G$), and random velocity ($\vec{V}_R$), received through the last sink update, which will be discussed in Section 7.3.6. The group velocity ($\vec{V}_G$) is multiplied by the time ($t$) elapsed since the time that A receives the last adaptive update. The displacement due to group mobility ($\vec{V}_G \cdot t$), combined with the last known location (L) of the sink node yields the center (S) of the estimated sink area. The random
velocity ($\vec{V}_R$) of the sink node multiplied by the time ($t$) elapsed since the last update yields the sink random mobility ($\vec{V}_R \cdot t$), or the radius of the estimated sink area.

After computing the estimated sink area, the source node constructs a $60^\circ$ forwarding area focused on the estimated sink area center, and also splits it into $1 \leq Q \leq 8$ sectors with a central angle of $\frac{60}{Q}$, as shown in Figure 7.5. $Q$ is specified at runtime and it is determined through evaluation that more than 8 sectors results in significant communication overhead in comparison with any accuracy gains. Each sector has an inner boundary, $l_i c$ (clockwise) or $l_i a$ (counter-clockwise), and outer boundary, $l_{i+1} c$ or $l_{i+1} a$, with respect to the source-estimated sink area center line $l_0$.

As shown in Figure 7.6, the slopes of sector boundaries are first computed from $l_0$ to $l_{[\frac{Q}{2}]i c}$ sequentially in a clockwise manner, and then from $l_0$ to $l_{[\frac{Q}{2}]i a}$ sequentially in a counter-clockwise manner.

Denoting locations of the source and estimated sink center as $(x_A, y_A)$ and $(x_B, y_B)$,
respectively, the slope of \( l_0 \) (or line \( AB \)) can be computed as:

\[
\tan(\alpha) = \frac{|y_A - y_B|}{|x_A - x_B|}
\]  

(7.1)

The slope of \( l_0 \) is then used in combination with the sector central angle \( \theta \) to compute \( \tan(\gamma) \), the slope of \( l_{1c} \). Since \( 1 \leq Q \leq 8 \), and hence there is a fixed, small number of possible \( \theta \) values, the value of \( \tan(\theta) \) can be obtained from a table lookup, rather than using the limited computation resources of small sensor devices.

\[
\tan(\gamma) = \tan(\alpha - \theta) = \frac{\tan(\alpha) - \tan(\theta)}{1 + \tan(\alpha)\tan(\theta)}
\]

(7.2)

Using Equation 7.2, for a given sector boundary \( l_{ic} \) with known slope \( \tan(\alpha) \), the slope for the adjacent sector boundary \( l_{i+1c} \) can be calculated as \( \tan(\gamma) \).

\[\text{(a) Calculating the slope of clockwise boundaries}
\]

\[\text{(b) Calculating the slope of counter-clockwise boundaries}
\]

**Figure 7.6:** The slope of a boundary \( l_{1j} \), where \( j \in \{c,a\} \), can be calculated using the known values \( \tan(\theta) \) and slope of \( l_0 \), \( \tan(\alpha) \). The slope of \( l_{2j} \) can then be calculated from the slope of \( l_{1j} \).

Once all the sector boundaries from \( l_{1c} \) to \( l_{\lfloor \frac{Q}{2} \rfloor_c} \) have been computed, the sector boundaries from \( l_{1a} \) to \( l_{\lfloor \frac{Q}{2} \rfloor_a} \) are computed sequentially in a counter-clockwise fashion. The process mirrors that used for sector boundaries \( l_{1c} \) to \( l_{\lfloor \frac{Q}{2} \rfloor_c} \). For example, using Equation 7.1 and
the sector central angle $\theta$, the slope of $l_{1a}$, or $\tan(\gamma)$ can be calculated as:

$$\tan(\gamma) = \frac{\tan(\alpha + \theta)}{1 - \tan(\alpha)\tan(\theta)}$$

With the boundaries determined for each sector, the source node then generates $N$ random locations, $8 \leq N \leq 64$, uniformly within the estimated sink area (dashed circle in Figure 7.7), to represent possible sink locations (white points). As with the number of sectors, it is determined during evaluation that using more than 64 locations incurs a great amount of computational overhead in comparison to a small increase in forwarding accuracy. For each generated location $R$, we compute which sector in which it is located by comparing the slope of the line that connects this location and the source $A$ (line $AR$) with that of sector boundaries. With the number of possible sink locations in each sector computed, a source node piggybacks this information in a data packet, together with locations of the source and estimated sink center. Compared with existing geographic forwarding-based protocol like GPSR [63], only $Q$ extra bytes are needed for forwarding to a mobile sink. In Algorithm 10, we summarize the initialization phase.

Algorithm 10 Initialization

Input The number of possible sink locations to predict $N$, the source location $A(x_A, y_A)$, the estimated sink center $S(x_S, y_S)$, and radius $r_S$ ($r_S = V_R \cdot t$ as shown in Figure 7.5).

Output The number of possible sink locations in each sector, $C_i$, $1 \leq i \leq Q$

1: cntLocations = 0;
2: while (cntLocations $< N$) do
3:     Generate random location $R(x_R, y_R)$ inside circle $(S, r_S)$;
4:     if ($R$ is in sector $i$) then
5:         $C_i$++;
6:     cntLocations++;
7: end if
8: end while
Figure 7.7: The source (A), generates 8 possible sink locations (white) in its estimated sink area (dashed circle) and determines the number of points in each sector before transmitting. Node F, the forwarder, creates a new estimated sink area (solid circle) and node A’s sectors. F generates new possible sink locations (gray) in overlapping areas of A’s and its own predictions, based on the number of points in A’s corresponding sectors. Remaining points (P1 and P2) are created uniformly within the estimated sink area until there are \( N \) possible sink locations.

7.3.5 SMC Prediction, Filtering, and Resampling

After a data packet is forwarded from a source to the next hop, three things happen:

1) Prediction: the \( N \) sink locations generated by the source is used to predict current sink locations again\(^1\). Also, the current node predicts sink locations based on its own knowledge;

2) Filtering: the current node makes use of both the previous node’s and its own sink location predictions to filter out impossible sink locations;

3) Resampling: the current node uses its own knowledge to generate new possible sink locations to replace those eliminated in the filtering phase.

Since the three phases are closely related, we explain them together. As shown in

---

\(^1\)The elapsed time between when the previous node predicts the sink location and when the current node needs to predict the sink location is the same as a single hop data forwarding time. It is usually a few milliseconds. Since the elapsed time is so small, the sink movement within such a period can be ignored. Therefore, an intermediate forwarding node can just use the \( N \) possible sink locations generated by a previous node to reconstruct, rather than predict, the current sink location.
Figure 7.7, when node F receives a data packet from node A, several items are reconstructed. From the information piggybacked in the data packet, node F determines the following: the 60° forwarding zone from node A, the 4 sectors in the forwarding zone, and also the 2/3/3/0 sink location distribution within the sectors. Node F also predicts possible sink locations (gray points), based on its own knowledge of the sink location as well as its group and random velocities received in the last sink update.

In the filtering process, the accumulated sink prediction, which is represented by the number of locations in each sector, is filtered using node F's prediction. The accumulated sink predictions are plotted within the solid circle in Figure 7.7. For instance, since sector 1 and 2 overlap with node F's prediction circle, the 3/3 location distribution is regenerated in the corresponding overlapping area between sectors and the solid circle. Also, since sector 0 has no overlapping with the circle predicted by node F, the 2 locations in this sector are filtered out. Since this filtering phase only regenerates 6 sink locations, 2 more locations are needed to form the new prediction. As shown in Figure 7.7, these new locations P1 and P2 are resampled uniformly from node F's prediction.

With a similar technique as presented in Section 7.3.4, 4 sectors can be formed at F towards the newly estimated sink center. Then, the location of F, the estimated sink center of F, and the number of locations in each sector are piggybacked again in the data packet and forwarded to the next hop.

The core of the filtering and resampling phases is summarized in Algorithm 11. One important aspect of this algorithm is that we do not directly generate the specified number \( (C_i) \) of locations within the overlapping area of each sector and the new estimation area \( (S, r_S) \), since that requires too much computation for sensor nodes. Instead, we generate
enough random locations (10 times the expected number $N$) within the circular area, and determine the sector in which they are located. We are aware that if an overlapping area of a sector and the new estimation area is very small, it may not get any location generated inside, even though its $C_i$ value is not zero. This does not have a noticeable impact on accuracy since only a tiny overlapping area is ignored when the total number of random locations is 10 times that of what is expected.

### Algorithm 11 Filtering and Resampling

**Input** The number of possible sink locations to predict $N$, the previous node location $A(x_A, y_A)$, the estimated sink center $S(x_S, y_S)$ and radius $r_S$ ($r_S = V_R \cdot t$ as shown in Figure 7.5) of the current node $F$, the number of possible sink locations in sector $i$ of node $A$, $C_i$, $1 \leq i \leq Q$.

**Output** Newly predicted $N$ sink locations for node $F$.

```
1: cntLocations = 0; cntSectorNodes = 0;
2: /* Filtering */
3: while (cntSectorNodes < $N \times 10$) do
4:     Generate random location $R(x_R, y_R)$ inside circle $(S, r_S)$;
5:     if ($R$ is in sector $i$ of node $A$) then
6:         $C_i$--;
7:         cntLocations++;
8:     end if
9:     cntSectorNodes++;
10: end while

/* Resampling */
11: while (cntLocations < $N$) do
12:     Generate random location $R(x_R, y_R)$ inside circle $(S, r_S)$;
13:     Mark $R$ as an estimated sink location;
14:     cntLocations++;
15: end while
```

### 7.3.6 Adaptive Update

The SMC Prediction module is updated with sink location, group and random mobilities in a time and space adaptive manner via the Adaptive Update module. The method used is similar to that in [10], where nodes notify each other of their location at a rate based on distance and relative velocity in order to reduce bandwidth and energy costs. In Sidewinder,
a sink node updates its location and mobility behavior according to Equation 7.4 so that all Limited Flooding initiated two hops away will be successful.

\[ |\vec{V}_R| \times t \geq 2r \]  

(7.4)

In Equation 7.4, \( \vec{V}_R \) refers to the sink random velocity, \( t \) is the time since the last sink update, and \( r \) is the radio range, which is empirically configured. With this time-adaptive update, all one-hop neighbors of the sink at its last update will be no more than two hops away from the sink before the sink updates again, allowing all Limited Flooding packets to reach the sink.

Bandwidth and energy are further reduced by decreasing the sink update frequency as distance to the sink increases. To achieve this, each node has a probability \( P \) for forwarding a sink update packet. The value of \( P \) for a given node is based on its number of hops from the sink, \( h \). If a node is \( h \) hops away from the sink, the probability for it to forward a sink update packet, \( P_h \), is computed as:

\[ P_h = \alpha^{h-1}, (0 < \alpha \leq 1) \]  

(7.5)

In Equation 7.5, \( \alpha \) is set as 1 for the first update, so that all nodes in the network have initial knowledge of sink location and movement behavior. After system initialization, \( \alpha \) is reduced to an empirical value (0.2 is used in our evaluation) so that nodes farther away from the sink receive fewer updates. Nodes farthest away from the sink need only a general idea of the sink location and movement behavior; nodes that forward the data have a more accurate picture of the sink location and movement. As data makes its way from the
network edge to the sink, more accurate sink information is used at each hop to precisely route data to the sink.

7.4 Evaluation

Sidewinder is implemented in TinyOS-2.x [51] with nesC [41] and evaluated in TOSSIM [98] using B-MAC [99]. Two mobility models are used in evaluation: Random Waypoint without pause time [135], and the Reference Point Group [4] mobility model. Though more advanced mobility models exist [137], we choose Random Waypoint and Reference Point Group since they are simple and apply to a large number of possible scenarios, ranging from flood tracking to movements of search and rescue teams [16]. When Random Waypoint mobility is used, 500 nodes are uniformly deployed in an area of 215m x 215m, with the radio range of 25m. When Reference Point Group mobility is used, the deployment region is increased to 2000m x 2000m to allow for group movement. Also for Reference Point Group mobility, the group radius is set to 120m to maintain the same node density as Random Waypoint and each node's random movement is set to the maximum group speed. In both mobility cases, 3 sources are randomly chosen to report sampled readings to the same sink, at the frequency of one packet per second. In Sidewinder, the Mobility Monitor module beacons location information every 10s and the maximum backoff window in zone-based forwarding competition is 128ms. In GF, each node beacons every 1.5s and a neighbor table entry expires every 6.7s, which is identical to the evaluation of GF in [63]. We also replace the neighbor table maintenance in GF with a 60° zone-based forwarding strategy [14] [49] [77] [144], but without SMC prediction, and call the modified protocol Beaconless GF. In GF
and Beaconless GF, a sink node floods its location every 10s. We repeat each evaluation 100 times and present the averaged results in Figures 7.8, 7.9 and 7.10, together with 90% confidence intervals.

![Packet Delivery Ratio](image)

**Figure 7.8: End-to-End Packet Delivery Ratio**

As shown in Figure 7.8, Sidewinder significantly outperforms GF and Beaconless GF when nodes are mobile. For instance, when node speed is 20m/s, Sidewinder achieves 92% packet delivery ratio in group mobility, which is 52% higher than that of Beaconless GF and 42% higher than that of GF. Sidewinder also exhibits an 82% packet delivery ratio in random mobility, which is 20% higher than that of Beaconless GF and 72% than that of GF. This is because Sidewinder’s SMC prediction continuously corrects the data forwarding direction towards the mobile sink and the zone-based forwarding tolerates excessive topology changes due to mobility. GF does not have any of these mobility-aiding techniques and Beaconless GF does not have the SMC prediction technique. For the same reasons, Sidewinder always maintains a high packet delivery ratio, ≥ 80, while GF and Beaconless GF suffer from increasing node mobility.

In Figure 7.8, we also observe that Beaconless GF achieves a higher packet delivery
ratio than GF in random but not group mobility. Beaconless GF achieves 50% higher performance than GF in random mobility, because GF's neighbor table maintenance is negatively impacted by excessive topology changes. These topology changes are tolerated by the zone-based forwarding in Beaconless GF. In group mobility, GF achieves 10% higher performance than Beaconless GF. This is because the relative movement between nodes is small in comparison with a random mobility model with the same average node velocity. However, with group mobility, the 60° forwarding area constraint in Beaconless GF eliminates possible routing paths that GF can find. This also explains why Sidewinder achieves less than a 100% packet delivery ratio when there is mobility. We plan to address this issue in future, e.g., by borrowing the wisdom of face routing [63].

We also measure the end-to-end time delay and energy consumption per successfully delivered data byte to the sink, and present results in Figure 7.9 and 7.10. Figure 7.9 demonstrates that Sidewinder and Beaconless GF achieve similar but much lower time delay than GF, especially in random mobility. This is due to routing loops in GF. Such loops are caused by the comparative random movement between neighboring nodes, and
that also explains why the measured time delay in the random mobility case is significantly higher than that in the group mobility case. Since energy conservation is of utmost concern in wireless sensor networks, we depict Figure 7.10 as the amount of overhead used by Sidewinder, Beaconless GF, and GF. Increases in number of radio transmissions, packet sizes, and computation will result in increased energy consumption. Figure 7.10 demonstrates that Sidewinder and Beaconless GF consume similar but much less energy than GF, which is due to two reasons. First, GF expends significant energy on high frequency beacons for updating neighbor tables. Second, GF wastes more energy on packets that fail to be delivered to the mobile sink. GF also demonstrates better energy efficiency in group than random mobility, since multi-hop routing failures are more prevalent with the increased number of topology changes in random mobility.

7.5 Conclusion

This chapter presents Sidewinder, a novel protocol for in-situ data collection in mobile wireless sensor networks. We show through quantitative evaluation that traditional ad hoc
and wireless sensor routing solutions fail in highly mobile environments. Thus, Sidewinder addresses the issues of highly dynamic network topologies and static route failures with Sequential Monte Carlo prediction of sink locations. As a data packet makes its way from source to sink, the sink location prediction computed at each node is combined and updated with each successive hop, increasing prediction accuracy. We integrate this forwarding mechanism into Sidewinder using a one-dimensional clustering technique that preserves sink location prediction accuracy while minimizing bandwidth and energy overhead. Our performance evaluation in TOSSIM demonstrates that Sidewinder significantly outperforms state-of-the-art solutions in packet delivery ratio, time delay, and energy efficiency.
Chapter 8

Conclusion and Future Work

To conclude this dissertation, in Section 8.1, we revisit the contributions of the presented work, and in Section 8.2, we propose potential extensions to be explored in the future.

8.1 Contributions Revisited

While first generation wireless sensor network deployments focus on best effort sensing and solving fundamental hardware, computation, and communication issues, current sensor network deployments are increasingly focused on sensing performance. Applications for military surveillance [47], vehicular traffic congestion control [28], and elderly patient monitoring [109] are deployed on wireless sensor platforms and require strict performance guarantees in terms of accuracy and system lifetime. To meet these requirements, methods are needed to learn the capabilities of each specific deployment, capturing the sensing diversity present among heterogenous sensors, and using the learned sensing diversity to choose the most accurate and energy efficient sensors.

We address challenges in three major areas towards exploiting sensing diversity to pro-
vide high accuracy and significant energy savings for performance critical applications. First, using machine learning, we must be able to explore and capture sensing diversity present in a sensor deployment. To provide high accuracy and energy savings, we investigate different machine learning techniques and evaluate their suitability for low power sensor networks. We use learned sensing capability to locate the most helpful sensors and enable sensor collaboration only when needed. Second, we address sensing diversity in distributed sensor networks, such as deployments for detection and monitoring of vehicular traffic or natural disasters. We provide generic solutions which can work with a wide range of sensor modalities, machine learning methods, and deployments. Our solutions easily adapt to environmental dynamics during runtime. We also address the challenge of routing detection or classification decisions in a mobile environment. Third, we address sensing diversity in body sensor networks, such as in deployments for personal health care and physical fitness assessment. We provide a user friendly and practical solution which is portable, accurate, and computationally lightweight. Our solutions easily adapt to the dynamics present in body sensor networks and enable sharing amongst neighboring BSNs to significantly improve accuracy and battery life.

We address these major challenges through the following contributions:

- **Watchdog.** Through quantitative study, we show that traditional approaches to event detection have difficulty meeting such requirements. Specifically, they cannot explore the detection capability of a deployed system and choose the right sensors, homogeneous or heterogeneous, to meet user specified detection accuracy. They also cannot dynamically adapt the detection capability to runtime observations to save
energy. Therefore, we are motivated to propose Watchdog, a modality-agnostic event detection framework that clusters the right sensors to meet user specified detection accuracy during runtime while significantly reducing energy consumption. We implement several machine learning techniques with which Watchdog can use to meet user requirements for event detection and show how these methods can perform in an online and adaptive manner. Through evaluation with vehicle detection trace data and a building traffic monitoring testbed of IRIS motes, we demonstrate the superior performance of Watchdog over existing solutions in terms of meeting user specified detection accuracy, energy savings, and environmental adaptability.

• **Wolfpack.** We are among the first to explore the impact of sensing diversity on sensor collaboration, exploit diversity to meet user specified accuracy requirements (confident sensing), and apply diversity exploitation for confident sensing coverage. We show that our diversity-exploiting confident coverage problem is NP-hard for any specific deployment and present a practical solution, Wolfpack. Through a greedy, distributed, and iterative sensor collaboration approach, Wolfpack maximizes a specific deployment’s capability to meet user detection requirements and save energy by powering off unneeded nodes. Using real vehicle detection trace data, we demonstrate that Wolfpack provides confident event detection coverage for 30% more detection locations while using 20% less energy than a state of the art approach.

• **PBN: Towards Practical Body Sensor Networks.** Most activity recognition deployments and applications are challenged to provide personal control and practical functionality for everyday use. We argue that activity recognition for mobile devices must
meet several goals in order to provide a practical solution: user friendly hardware and software, accurate and efficient classification, and reduced reliance on ground truth.

To meet these challenges, we present PBN: Practical Body Networking. Through the unification of TinyOS motes and Android smartphones, we combine the sensing power of on-body wireless sensors with the additional sensing power, computational resources, and user-friendly interface of an Android smartphone. We provide an accurate and efficient classification approach through the use of ensemble learning. We explore the properties of different sensors and sensor data to further improve classification efficiency and reduce reliance on user annotated ground truth. We evaluate our PBN system with multiple subjects over a two week period and demonstrate that the system is easy to use, accurate, and appropriate for mobile devices.

- **Remora.** In many Body Sensor Network (BSN) applications, such as activity recognition for assisted living residents or physical fitness assessment of a sports team, users spend a significant amount of time with one another while performing many of the same activities. We exploit this physical proximity with Remora, a smartphone-based Body Sensor Network activity recognition system which shares sensing resources among neighboring BSNs. Compared to individual BSNs, Remora resource sharing provides increased accuracy and significant energy savings. To increase classification accuracy, Remora BSNs share sensors by overhearing neighbors' sensor data transmissions. When sharing, fewer on-body sensors are needed to achieve high accuracy, resulting in energy savings by turning off unneeded sensors. To save phone energy, neighboring BSNs share classifiers: only one classifier is active at a time classifying
activities for all neighbors. Remora addresses three major challenges of sharing with physical neighbors: 1) Sharing only when the energy benefit outweighs the cost, 2) Finding and utilizing the shared sensors and classifiers which produce the best combination of accuracy improvement and energy savings, and 3) Providing a lightweight and collaborative classification approach, without the use of a backend server, which adapts to the dynamics of available neighbors. In a two week evaluation with 6 subjects, we show that sharing provides up to a 30% accuracy improvement for BSNs with poor individual performance while extending phone battery lifetime by up to 65%.

- **Sidewinder.** We demonstrate through quantitative study that traditional approaches to routing in mobile environments do not work well due to volatile topology changes. Consequently, we propose Sidewinder, a predictive data forwarding protocol for mobile wireless sensor networks. Like a heat-seeking missile, data packets are guided towards a sink node with increasing accuracy as packets approach the sink. Different from conventional sensor network routing protocols, Sidewinder continuously predicts the current sink location based on distributed knowledge of sink mobility among nodes in a multi-hop routing process. Moreover, the continuous sink estimation is scaled and adjusted to perform with resource-constrained wireless sensors. Our design is implemented with nesC and evaluated in TOSSIM. The performance evaluation demonstrates that Sidewinder significantly outperforms state-of-the-art solutions in packet delivery ratio, time delay, and energy efficiency.

Our work completed and proposed provides key insights and contributions for meeting
stringent accuracy and lifetime requirements in performance critical applications. With our work in distributed sensor networks, we demonstrate that sensing diversity, the sensing capability differences among heterogeneous and homogenous sensors, is an asset to be exploited to meet application requirements. With Watchdog and Wolfpack, we demonstrate that choosing the right sensors as well as when and how to collaborate sensors yields a dramatic impact on system performance. We also demonstrate the effectiveness in using node movement prediction to significantly increase packet delivery rates in mobile sensor network deployments. With body sensor networks, we show that lightweight and energy efficient solutions to activity classification can provide high accuracy by finding and utilizing the most helpful sensing resources. Through sharing sensors and classifiers, neighboring body sensor networks can collaborate to provide significant accuracy and energy saving gains. All of the work in this dissertation maximizes the capability of available sensing, computation, and communication resources to meet application or user requirements while extending system lifetime.

8.2 Future Work

In this section, we suggest potential areas of future work for both the existing work we have proposed as well as for wireless sensor networks and mobile computing in general.

Distributed Sensor Networks. While our work with Wolfpack and Watchdog finds and uses the best sensors to meet user accuracy requirements while saving energy, it also presents an energy load balancing problem. The most accurate sensors and nodes may see their battery energy depleted well before the remainder of the deployment. We propose
to extend our diversity exploiting clustering approach to consider deployment lifetime as a whole rather than just reducing the number of nodes used. By extending the Wolfpack solution, we aim for nodes to form clusters based on both sensing capability and remaining battery energy. Nodes are periodically reclustered not only due to environmental dynamics but also energy consumption to ensure energy fairness.

We also propose several improvements to our Sidewinder data forwarding protocol for mobile wireless sensor networks. We first propose to further improve node movement and sink location prediction through the use of a Kalman filter. While our current approach only considers the latest individual node movements in forming location predictions, a Kalman filter will allow more accurate predictions by considering node movement history. We also intend to improve the quality of the simulation by introducing radio irregularity and localization errors.

**Body Sensor Networks.** We propose several extensions to both our PBN and Remora activity recognition approaches. First, we intend to provide a more extensive usability study with a diverse array of subjects. With more subjects and activities, we will be able to identify new research challenges for body sensor networks, such as finding certain activities or users which are subject to high packet loss and provide high classification accuracy in the face of such loss. We also propose to extend our sensor and classifier sharing approach to the cloud to further improve accuracy as well as investigate how to share classifiers among neighboring BSNs when such neighbors are performing different activities.

**Adaptive Mobile Infrastructure.** The explosion of cloud computing has mirrored that of mobile devices, prompting many mobile application developers to offload data storage and energy-intensive computation to the cloud. However, the trend towards using mobile
phones in conjunction with cloud computing raises many concerns which have not been fully investigated. First, the exact energy and computation tradeoffs for various phone-cloud applications have not been fully addressed. While the authors of many mobile application papers argue that cloud computing allows for the offloading of computationally intensive tasks to more powerful machines, significant energy may be expended in wireless communication. Second, cloud computing is notorious for its reliability issues, and furthermore, may not be available in the case of a weak or nonexistent wireless signal. Lastly, many mobile device users may be concerned about the privacy implications of cloud computing and data storage and may not wish to use such services to handle confidential data.

In light of these issues, we propose to explore the tradeoffs between mobile and cloud computing, considering energy, computation, reliability, and privacy concerns. We intend to provide a framework for a mobile phone operating system that is designed to handle various configurations of phone and cloud-based computation and data storage. Application and user preferences will be considered to automatically adjust the usage profile of both phone and remote resources.

**Data Mining Meets Physical Sensing.** Until recently, the internet and physical domains have been quite separate, with most internet users having anonymous identities and sharing very little personal information. With the advent of faster internet connections and cheaply available internet data storage, every individual is leaving behind a treasure trove of information, ranging from social networking updates to blog posts and photo albums. Combining the extensive information available on the internet with physical world sensing will usher in a new era of applications in augmented reality, context awareness, and law enforcement. By determining internet data patterns and combining those inferences
with sensors in the real world, future applications will exhibit greater performance while relying on the same or fewer numbers of sensors as state of the art sensor network applications. We propose to investigate various internet data mining techniques and combine them with sensor system applications for context awareness and augmented reality to provide higher application fidelity while using fewer energy and computation constrained mobile and embedded sensor devices.
Bibliography


Matthew Alexander Keally

Matthew Keally received his Bachelor of Science and Master of Science in Computer Science from the College of William and Mary in 2006 and 2008, respectively. He began work on his Doctor of Philosophy in Computer Science in 2008. His research interests are in wireless sensor networks, cyber-physical systems, as well as mobile computing. His recent research focuses on developing systems and protocols for mobile devices which address the energy and computational limitations inherent in such devices while ensuring that user and application requirements are met.