5-2018

Energy Efficient Downstream Communication in Wireless Sensor Networks

Xiaoyang Zhong
Purdue University

ENERGY EFFICIENT DOWNSTREAM COMMUNICATION IN WIRELESS SENSOR NETWORKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Xiaoyang Zhong

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2018

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Yao Liang, Co-Chair

Department of Computer and Information Science

Dr. Yu C. Hu, Co-Chair

Department of Electrical and Computer Engineering

Dr. Fengguang Song

Department of Computer and Information Science

Dr. Ninghui Li

Department of Computer Science

Dr. He Wang

Department of Computer Science

Approved by:

Dr. Voicu Popescu by Dr. William J. Gorman

Head of Graduate Program

To my parents Rongliang and Xiangji, my sister Xiaodan, and my brother Xiaoming, for their love, understanding, and support.

ACKNOWLEDGMENTS

First, I would like to express my most sincere gratitude to my adivsor, Professor Yao Liang, for his mentorship and unconditional support during my doctoral studies. He has shaped my thinking and research attitudes, and taught me how to research and how to be a researcher. His instructions and encouragement made this dissertation possible.

I would also like to thank Professor Yu C. Hu for serving as my co-chair in my advisory committee, as well as other committee members Professor Ninghui Li, Professor Fengguang Song, and Professor He Wang. Their insights, discussions, and challenging questions have significantly enhanced many aspects of this dissertation.

I am also very grateful to our collaborators from the University of Pittsburgh, Professor Xu Liang, German Villalba, Thomas A. Slater, and Fernando Plaza. They have significantly contributed to this dissertation through their feedbacks and strong support in the field deployment and maintenance of sensor nodes.

I would also like to thank my colleagues and friends Yimei Li, Miguel Navarro, and Cong Qi for their help, discussions, support, and encouragements during my PhD studies.

Finally, I would like to thank my family for their unconditional support, care, understanding, and love for all the years.

TABLE OF CONTENTS

Page

LIST OF TABLES viii

LIST OF FIGURES ix

ABSTRACT xii

1 INTRODUCTION 1
1.1 Motivation 1
1.2 Downstream Communication in WSNs 3
1.2.1 Small Data Dissemination 3
1.2.2 Bulk Data Dissemination 4
1.3 Major Contributions 5
1.3.1 Scalable Downward Unicast Routing 5
1.3.2 Mobile Code Dissemination Tool 6
1.3.3 Network Analysis and Benchmarking 6
1.4 Organization 7

2 BACKGROUND 8
2.1 TinyOS 8
2.2 Drip 9
2.3 RPL 9
2.4 Deluge 10

3 SCALABLE DOWNWARD ROUTING FOR WIRELESS SENSOR NETWORKS 11
3.1 Related Works 13
3.2 OSR Design 15
3.2.1 Adaptive Bloom Filter for Path Encoding 15
3.2.2 Direct Child Set 18
3.2.3 Opportunistic Routing 20
3.2.4 Downward Routing Decision 22
3.2.5 OSR Optimizations 25
3.3 Analytical Scalability 26
3.3.1 Scalability of RPL 27
3.3.2 Scalability of ORPL and OSR 28

4 OSR IMPLEMENTATION AND EVALUATION 34
4.1 OSR Implementation 34

Page

4.2 Evaluation Methodology and Setup . . . 37
4.3 Evaluation in Cooja . . . 38
4.4 Evaluation in Indriya . . . 41
4.4.1 Comparison against RPL . . . 42
4.4.2 Comparison against Drip . . . 43
4.5 Evaluation in TOSSIM . . . 46
4.5.1 Normalized Transmission Cost . . . 49
4.5.2 The Size of the Direct Child Set . . . 51
4.5.3 Distribution of Opportunistic Routing . . . 53
4.5.4 The Effect of Maximum Bloom Filter Size . . . 54
4.6 A Case Study in ASWP WSN Testbed . . . 54
4.6.1 Deployment and Application . . . 55
4.6.2 Protocol Evaluation . . . 57
4.7 Discussion . . . 60

5 MOBILE CODE DISSEMINATION FOR WIRELESS SENSOR NETWORKS . . . 63
5.1 Related Work . . . 65
5.2 Design of MobileDeluge . . . 67
5.2.1 MobileDeluge Outline . . . 68
5.2.2 Subset Reprogramming . . . 68
5.3 Evaluation . . . 74
5.3.1 Lab Experiments . . . 75
5.3.2 Testbed Experience . . . 77

6 NETWORK DYNAMICS AND BENCHMARKING . . . 79
6.1 Related Work . . . 79
6.2 Data Source . . . 80
6.3 Network Dynamics . . . 82
6.3.1 Link Level Characteristics . . . 82
6.3.2 Topological Characteristics . . . 87
6.3.3 Temporal Characteristics . . . 93
6.4 Benchmark Data Suite . . . 94
6.4.1 Data Generation . . . 94
6.4.2 Fill Missing Paths . . . 95
6.4.3 Data Characteristics . . . 98

7 SUMMARY AND FUTURE WORK . . . 100
7.1 Summary . . . 100
7.2 Future Work . . . 102

REFERENCES . . . 103

APPENDIX: BENCHMARK DATA FORMAT . . . 111

VITA 113

LIST OF TABLES

Table Page

3.1 Notations for Analytical Scalability 27

4.1 Comparison of RAM Sizes for TelosB Platform 39

4.2 Scalability Comparison on Quasi Linear Network 40

4.3 Comparison between OSR and TinyRPL on Low Power Indriya Testbed with 47 Nodes 42

4.4 Comparison between OSR and Drip on Low Power Indriya Testbed with 95 Nodes 45

4.5 Statistics of Multicasts for OSR and OSR_{oh} in Indriya 46

4.6 OSR Overall Performance in TOSSIM Simulation 48

4.7 Detailed Statistics of OSR in TOSSIM Simulation 49

4.8 OSR Performance with Different Setups of *MAX_BFLT_LEN* in TOSSIM Simulation 55

4.9 Performance of OSR in ASWP Testbed 58

5.1 Performance of Deluge with LPL and Always-on 64

5.2 Program Image Size 75

5.3 Comparison of Deluge and MobileDeluge for Single Node 76

5.4 Comparison of Deluge (3 Nodes) and MobileDeluge (3 Out Of 5 Nodes) . . 77

5.5 Statistics of Reprogramming in the Field 78

6.1 Mote Radio Configurations 85

6.2 Overall Statistics of the Benchmark Datasets 98

A.1 Data Format of the Benchmark Dataset 112

LIST OF FIGURES

Figure Page

3.1 A conceptual illustration of downward packet delivery with OSR versus RPL/ORPL/CBFR. 13

3.2 The false positive rate of adaptive Bloom filter and the corresponding space saving. Path length is H hops; Bloom filter length m is based on (3.2); the number of hash functions $k = 3$; $MAX\_BFLT\_LEN = 40$ bytes. 17

3.3 Node distribution on the number of direct children for the Indriya testbed with 95 available nodes. 19

3.4 Illustration examples of (a) passive opportunistic routing and (b) active opportunistic routing in OSR. The source-route is marked as circles with double line border. 22

3.5 Upper bound of the neighborhood size (N_{nb}) with varies number of hash functions k, given the coefficient upper bound of 16 bits. 30

3.6 Upper bound of the neighborhood size (N_{nb}) for varies direct child set sizes (N_{ch}) given the coefficient upper bound being 2 bytes (16 bits). (a) varies number of functions k; (b) $k = 2, 3$. 33

4.1 OSR packet format. 35

4.2 OSR implementation with CTP/CTP+EER. *Path Encoding*, *Send*, and *OsrPacket* are for the sink side only. *Receive* is for the node side only. . . . 36

4.3 The illustration of the quasi linear topology. Root is at the left end. 39

4.4 Downward PDR for linear topology based on downward path length in the first 25 hops. 41

4.5 Nodes distribution for low power WSN test of CTP+Drip and CTP+OSR in Indriya. 44

4.6 Downsward packet delivery rate distributed on downsward path length. . . 48

4.7 Normalized transmission cost distributed on downward path length. 50

4.8 Average direct child set size distributed on node's average path length. . . 52

Figure Page

4.9 Average number of multicast occurrences distributed on node's average path length. 52

4.10 The normalized number of multicast occurrences distributed on downward path length. 53

4.11 The number of opportunistic routing occurrences distributed on downward path length. 54

4.12 Node locations at the ASWP WSN testbed as of August 2017. MicaZ nodes are marked as blue, IRIS nodes are marked as red, and TelosB nodes are marked as yellow. 56

4.13 Major components of the ASWP application. 57

4.14 Average direct child set size distributed on node's average distance in ASWP testbed. 59

4.15 Downward packet delivery ratio distributed on node's average distance in ASWP testbed. 59

5.1 MobileDeluge, a hand-held mobile code dissemination tool. 66

5.2 MobileDeluge packet structure. 69

5.3 Finite state machines of MobileDeluge control logic. 71

5.4 Start phase message exchange. 72

5.5 Code structures of Deluge and MobileDeluge. 73

5.6 An illustration of MobileDeluge gateway interface, showing the platforms and application versions of target nodes. 74

6.1 Overall link performance. 84

6.2 Distribution of noise floor and link types. 86

6.3 Link asymmetry based on RSSI for the links between the same types of nodes. 86

6.4 Link asymmetry based on RSSI for different link types. 87

6.5 Link asymmetry based on link PRR_retx for different link types. 88

6.6 Node distribution on their average distance to the sink. 89

6.7 CDF of node degrees. 89

6.8 Spatial distribution of the average parent set size of the nodes at the ASWP testbed. The larger the circle, the larger the parent set. 91

Figure Page

6.9 CDF of the number of links per cycle. 92

6.10 CDF of the link entropy and the node entropy. 92

6.11 Spatial distribution of the number of forwarded packets for each node at the ASWP testbed. The larger the circle, the more concentration of the traffic. Sink's two-hop neighbors have forwarded all the traffic and are shown in solid green dots. 93

6.12 Temporal characteristics. 94

6.13 Benchmark data generation process. 96

6.14 Statistics of the original data, the benchmark data with loopy packets (BM), and the benchmark data without loopy packets (BM_No_Loop). . . 99

ABSTRACT

Zhong, Xiaoyang Ph.D., Purdue University, May 2018. Energy Efficient Downstream Communication in Wireless Sensor Networks. Major Professor: Liang, Yao.

This dissertation studies the problem of energy efficient downstream communication in Wireless Sensor Networks (WSNs). First, we present the Opportunistic Source Routing (OSR), a scalable, reliable, and energy-efficient downward routing protocol for individual node actuation in data collection WSNs. OSR introduces opportunistic routing into traditional source routing based on the parent set of a node's upward routing in data collection, significantly addressing the drastic link dynamics in low-power and lossy WSNs. We devise a novel adaptive Bloom filter mechanism to effectively and efficiently encode a downward source-route in OSR, which enables a significant reduction of the length of source-route field in the packet header. OSR is scalable to very large-size WSN deployments, since each resource-constrained node in the network stores only the set of its direct children. The probabilistic nature of the Bloom filter passively explores opportunistic routing. Upon a delivery failure at any hop along the downward path, OSR actively performs opportunistic routing to bypass the obsolete/bad link. The evaluations in both simulations and real-world testbed experiments demonstrate that OSR significantly outperforms the existing approaches in scalability, reliability, and energy efficiency. Secondly, we propose a mobile code dissemination tool for heterogeneous WSN deployments operating on low power links. The evaluation in lab experiment and a real world WSN testbed shows how our tool reduces the laborious work to reprogram nodes for updating the application. Finally, we present an empirical study of the network dynamics of an out-door heterogeneous WSN deployment and devise a benchmark data suite. The network dynamics analysis includes link level characteristics, topological characteristics, and temporal

characteristics. The unique features of the benchmark data suite include the full path information and our approach to fill the missing paths based on the principle of the routing protocol.

1 INTRODUCTION

1.1 Motivation

Wireless Sensor Networks (WSNs) are composed of a large number of tiny, low-cost, low-power, and multifunctional sensor nodes that are capable of sensing, data processing, and radio communication. Once deployed, sensor nodes work collaboratively to sample environmental phenomenon and transmit the data to one or more sink nodes in a multihop manner. Due to its low-cost and easy to use, WSNs have been increasingly adopted in various application areas such as environmental monitoring [1], structural health monitoring [2], smart buildings [3], smart cities [4], precision agriculture [5], and e-health systems [6]. Sensor nodes are notable resource constraint devices due to its small size, with limited processor capacity, memory, communication bandwidth, and energy supply. The resource limitations require low-overhead design on both the hardware and software of WSNs.

In typical WSNs applications, sensor nodes periodically sample and transmit the environmental data upwards to one or multiple sinks, which can be characterized as *upstream* communication. Many approaches have been proposed in the literature to address and optimize the upward data delivery, representative protocols include the Collection Tree Protocol (CTP) [7] and the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [8]. On the other hand, downstream communication from sink to individual sensor/actuator nodes is also essential in WSN deployments, for instance, to reconfigure network parameters (e.g., sampling rate), to query reports from specific nodes, or to reprogram the nodes to run new applications. This type of communication is characterized as *downstream*. Popular downstream protocols include Drip [9], Glossy [10], RPL [8], and Deluge [11].

Since data collection is the basic task for WSNs, it is very important to understand the network behavior of multihop WSN deployment during operation. Although many efforts have been made in the last decades on the WSN deployments and many experiences and guidelines have been reported [12–15], the deployments are usually short term and homogeneous. Specifically, very few of the studies focus on the routing dynamics of the network given that routing is the critical component of any multihop WSN deployment. Thus, it is a necessity to provide the experiences learned from the long-term heterogeneous WSN deployment and benchmark the routing dynamics in such deployment.

The evolving WSNs introduce the following distinctive challenges to the design of communication mechanisms besides the resource limitations on individual sensor nodes:

- Large Scale: The low cost of WSNs makes it feasible to obtain high resolution measurements of the area of interest using a large number of sensor nodes. On the other hand, covering a large monitoring area requires a large amount of nodes since nodes' effective communicating range is usually in the order of tens of meters.

- Low Power: WSNs usually operates under low power settings (i.e., duty cycled) to achieve long lifetime. Nodes spend most of the time sleeping and wake up periodically to perform sensing task, reducing the energy consumption to the minimum.

- Heterogeneous: The nodes in a WSN deployment may vary in both hardware and software configurations based on their locations and roles in the network.

Compared to optimized upward routing, downstream communication is significantly less studied. Popular downstream protocols are usually broadcast based and only allow the sink to disseminate packets to the entire network. Broadcast limits the scalability and energy efficiency of those protocols. The lack of addressing individual

node(s) in those dissemination protocols makes them very inefficient or even infeasible for downstream node control in real-world heterogeneous WSNs. Moveover, low power settings may significantly affect the performances of the popular downstream protocols. Thus, new designs for energy efficient one-to-one or one-to-many downstream communication are necessary for low-power large-scale heterogeneous WSN deployments.

The standard RPL, the IPv6 routing protocol for low-power and lossy networks [8], offers the capacity of downward routing through its either storing mode or non-storing mode. However, it has been found that RPL has several significant flaws in its downward point-to-multipoint communication (e.g., [16–24]). RPL essentially suffers from the severe scalability problem for downward routing [20–23]. Moreover, it seems that RPL might not effectively fix any unreachable failure in downward routing due to wireless link dynamics. Although recent approaches such as ORPL [25] and CBFR [26] attempted to address the scalability issue of downward routing, these improvements are limited for highly resource-constrained wireless devices (see [21], for example).

1.2 Downstream Communication in WSNs

In general, downstream communication in WSNs can be classified in two major categories: small data dissemination and bulk data dissemination. Small data dissemination usually targets to parameter reconfiguration or data query, whereas bulk data dissemination is mainly used for node reprogramming.

1.2.1 Small Data Dissemination

Classical approaches to small data dissemination are based on controlled flooding. Once new data are available at the sink, the data are broadcasted throughout the network. Existing approaches mainly differ in the way how broadcasts are controlled at individual nodes to achieve fast and energy-efficient data dissemination [9, 10, 27–

31]. The major drawback of this type of work is that the data is disseminated to the entire network, which is unsuitable for large-scale heterogeneous WSNs.

Recently downward unicast routing approaches start emerging in the academic area, enabling data dissemination for heterogeneous networks [8, 25, 26]. The differences from the upward routing impose new challenges to downward unicast routing. Upward routing is usually many-to-one, hence a gradient based routing metric can be efficiently computed at each node for making routing decision. In the end all the upstream packets would be merged to the node(s) with the lowest gradient, which is the sink(s). In contrast, gradient based forwarding mechanism could not be directly applied to downward unicast routing. Therefore, locating the next hop is an essential problem. Existing approaches [8,25,26] either rely on the traditional source routing or store the information of the entire subtree rooted at each intermediate node, suffering from severe scalability problem.

1.2.2 Bulk Data Dissemination

Over-the-air programming is the major purpose of bulk data dissemination, in which the sensor nodes can be reprogrammed wirelessly without physical contact. Many approaches have been proposed, such as Deluge [11], XNP [32], MOAP [33], and MNP [34]. However, most of them disseminate the new program to all the nodes in the network, failing to deal with heterogeneous WSNs. Moreover, they only work well on always on networks. In reality, WSN deployments are usually operates on low power link layer to achieve energy efficiency and long lifetime. In typical low power settings (e.g., BOX-MAC [35]), nodes spend most of the time sleeping, and wake up periodically to sense the radio channel for data reception. To transmit a packet, the sender node usually transmits a long preamble, until the receiver wakes up, before the packet can be delivered. Since application program data usually contain thousands of packets, low power link layer would significantly affect the performance of most of

the approaches. Thus, heterogeneous WSN deployments operating on low power link layer have introduced unique challenges to efficient bulk data dissemination.

1.3 Major Contributions

This dissertation studies the problem of energy efficient downstream communication in heterogeneous WSNs. We first address the downstream communication from the perspectives of small data dissemination and bulk data dissemination, then present an empirical study of a WSN deployment and devise a benchmark data suite.

1.3.1 Scalable Downward Unicast Routing

The problem of small data dissemination in heterogeneous WSNs is addressed by our newly developed OSR, the Opportunistic Source Routing, a scalable, reliable, and energy-efficient downward routing protocol for data collection WSNs. OSR introduces opportunistic routing into traditional source routing based on the parent set of a node's upward routing in data collection, significantly addressing the drastic link dynamics in low-power and lossy WSNs. We devise a novel adaptive Bloom filter mechanism to effectively and efficiently encode a downward source-route in OSR, which enables a significant reduction of the length of source-route field in the packet header. OSR is scalable to very large-size WSN deployments, since each resource-constrained node in the network stores only the set of its direct children. The probabilistic nature of the Bloom filter passively explores opportunistic routing. Upon a delivery failure at any hop along the downward path, OSR actively performs opportunistic routing to bypass the obsolete/bad link. We demonstrate the desirable scalability of OSR against the standard RPL downward routing. We evaluate the performance of OSR via both simulations and real-world testbed experiments, in comparison with the standard RPL (both storing mode and non-storing mode), ORPL, and the representative dissemination protocol Drip. Our results show that OSR significantly outperforms RPL and ORPL in scalability and reliability. OSR

also achieves significantly better energy efficiency compared to TinyRPL and Drip which are based on the same TinyOS platform as OSR implementation.

1.3.2 Mobile Code Dissemination Tool

We develop MobileDeluge, a novel hand-held mobile over-the-air mote reprogramming tool for outdoor heterogeneous WSN deployments operating on low power links. MobileDeluge builds a new control layer on top of Deluge. It enables and disables Deluge services on demand, allowing for the selection of a subset of motes as targets when initiating a reprograming task. It also disables the low power mode in the targets for fast dissemination of the new application image, which usually consists of thousands of packets. To avoid interference with the rest of the network, the bulk data is disseminated in a different radio channel. MobileDeluge works with the nodes within a one-hop range to avoid forwarding of the bulk code image over intermediate nodes for node energy conservation. The evaluation demonstrates that MobileDeluge has significantly reduced the time and labor required to update the application in the outdoor WSN testbed.

1.3.3 Network Analysis and Benchmarking

We present an empirical study of the network dynamics in an outdoor heterogeneous multihop WSN deployment, including the link level characteristics, topological characteristics, and temporal characteristics. We devise a benchmark data suite based on the data collected from the deployment during long-term operation. The main features of the benchmark includes the link information between heterogeneous hardware platforms and the complete topological information. Results show that asymmetric links are the majority in heterogeneous networks and the main cause is the hardware heterogeneity. Since the raw data is incomplete and dirty, we present our method to clean up the data and to fill the missing paths in each individual network topology. As a result, three datasets are generated to form the benchmark date suite: the cleaned

original data with missing paths, the benchmark data with supplement paths and loopy packets, and the benchmark data with supplement paths but without loopy packets.

1.4 Organization

This dissertation is organized as follows. Chapter 2 describes the background and the representative downstream protocols in detail. Chapter 3 describes the scalable downward routing protocol. The detailed evaluation of the protocol is presented in Chapter 4. Chapter 5 presents the design and evaluation of the mobile code dissemination tool. Chapter 6 presents the network dynamics analysis and benchmarking of an outdoor heterogeneous WSN deployment. Chapter 7 summarizes the current work and discusses the future directions.

2 BACKGROUND

This chapter presents the development environment that is used in this dissertation and the representative approaches on small data dissemination and bulk data dissemination.

2.1 TinyOS

TinyOS [36] is an open source operating system developed based on the nesC language [37] and is one of the most widely used WSN operating system that is found in 60% of WSN deployments [38, 39]. Unlike monolithic OSes, TinyOS provides a set of reusable components which are included as-needed in applications to provide efficient and extremely low-power operations. The operating system base is as small as 400 bytes in RAM. The components in a TinyOS program are connected through *interfaces*, which supports bidirectional interaction through *command* calls and *event* signals. Components can defer time consuming computations using *tasks*, which are executed by TinyOS scheduler on the background in a later time following a run-to-completion execution model, hence to increase system responsiveness. TinyOS provides the decent network simulator TOSSIM [40] to aid the development of WSNs applications. To achieve high fidelity simulation results, TOSSIM takes an environmental noise trace as input and creates accurate noise model for each simulated sensor node. A commonly used real world environmental noise trace is Meyer Heavy noise trace which was taken at the Meyer library at Stanford during heavy 802.11 activities [41, 42].

2.2 Drip

Drip [9] exploits the packet transmission algorithm of Trickle [43] and builds a concise transport-layer interface atop of it. The TinyOS implementation allows Drip to disseminate multiple types of message with distinct message identifiers. A component that uses Drip must specify the message identifier it wants to listen to.

Trickle uses a local "polite gossip" to maintain data consistency within the network. At each time interval, each node broadcasts its own data at a random point if it does not hear other nodes transmissions of the same data. If it hears a neighbor transmits the older data, it triggers every node up to date to disseminate the new data. If all neighbors broadcast the same data, Trickle doubles the packet time interval to a maximum of τ_h to achieve low transmission cost. If the network is injected with new data, the timer is reset to the shortest interval τ_l for fast information propagation. Due to periodic transmission, Trickle is resistant to network transience, loss, and disconnection, and easily handles network repopulation. Trickle has been standardized by IETF as RFC 6206, and has been used as the underlying retransmission scheme by many protocols, such as Drip, CTP, Deluge, and RPL.

2.3 RPL

RPL, the IPv6 routing protocol for low power and lossy networks [8], is a recent IETF standard routing protocol for LLNs that supports upward, downward, and point-to-point (P2P) traffic patterns. The upward routing shares many common principles with CTP, such as adaptive beacon interval based on Trickle timer and parent selection based on routing gradients. RPL supports two modes of downward routing, either through the entire subtree stored at each intermediate node's routing table (storing mode), or through the source-route specified at the root (non-storing mode). Both modes suffer from scalability problems. For both modes, each node sends destination adverstisement object (DAO) messages towards the sink following the upward routing in order for the parents/root to collect downward routing information.

The DAO packets may result in large overhead for downward routing. A recent and comprehensive survey of RPL can be found in [16].

2.4 Deluge

Deluge is the *de facto* default reprogramming protocol in TinyOS which is designed to disseminate the new application to the entire network. It uses ADV-REQ-DATA three way handshaking mechanism to ensure delivery reliability. In Deluge, a program image is divided into a set of fixed-size pages, whereas each page consists of a number of packets. Hence, a program image is distributed across the network in the form of hundreds/thousands of packets. A node in Deluge has three states: maintenance, request, and transmit. In maintenance state, Deluge broadcasts ADV packets following a Trickle timer to detect the inconsistency of program metadata among nodes. If a new version of program is detected, the node switches to request state and broadcasts REQ packets. If a node receives a request from neighbors for a new piece of data it has, it switches to transmit state and sends DATA packets containing the new image data to the neighbors. In request state, a node uses random delayed timer to broadcast requests to reduce collision. If all new data is received, the node switches to maintenance state. When a node finishes transmitting the new data, it switches to maintenance state. Deluge exploits spatial multiplexing to achieve fast dissemination in multihop WSNs.

In TinyOS based WSNs, Deluge uses Drip to disseminate control messages to the entire network for starting/stopping the new image distribution process.

3 SCALABLE DOWNWARD ROUTING FOR WIRELESS SENSOR NETWORKS

Recently downward unicast routing approaches start emerging in the academic area, enabling data dissemination for heterogeneous networks [8, 25, 26]. RPL is a recent IETF standard routing protocol that addresses unicast downward routing [8]. However, it has been found that RPL has several significant flaws in its downward point-to-multipoint communication (e.g., [16–24]). RPL essentially suffers from the severe scalability problem for downward routing [20–23]. In RPL storing mode, a node stores routing entries for all destinations in its subgraph/subtree, potentially suffering from severe scalability and reliability problem in large WSNs. On the other hand, RPL non-storing mode uses source routing [44] through the sink/root, which suffers from not only increased risk of packet fragmentation and thus increased battery power and network capacity consumption, but also the scalability issue of the possible length of route in a network, given constrained wireless layers, such as IEEE 802.15.4 with a maximum frame size of 127 bytes (including header) [45]. Moreover, it seems that RPL (non-storing mode) might not effectively fix any unreachable failure in downward routing due to wireless link dynamics. Although recent approaches such as ORPL [25] and CBFR [26] attempted to address the scalability issue of downward routing, these improvements are limited for highly resource-constrained wireless devices (see [21], for example). Indeed, it is increasingly urgent to systematically study scalable, reliable and resource-efficient WSN downward routing for emerging large-scale and resource-constrained WSN system.

Source routing includes the source-route information in the packet header to route packets from the source node to destination without building and maintaining routing tables at intermediate nodes. However, a direct application of source routing (e.g., RPL non-storing mode) to WSN downward routing is problematic. First, the

dynamic nature of WSNs significantly affects the reliability of the traditional source routing. The specified source-route of a downward packet may be obsolete and therefore unavailable when the packet arrives at an intermediate node due to wireless link dynamics. Second, the traditional source routing does not scale well in WSNs, because physical layer protocols of WSNs are designed to have a small frame size (e.g., IEEE 802.15.4 [45]) for energy efficiency. As the network diameter and hence the path length increases, containing the full source-route in a packet is inefficient and may even be infeasible. Thus, a desirable and practical source routing protocol in WSNs must simultaneously satisfy the requirements of reliability under the highly dynamic wireless communication environment and scalability for very large WSN deployments.

In this chapter, we present an Opportunistic Source Routing protocol, referred to as OSR, to achieve desirable scalability and reliability for heterogeneous WSN actuation. Our approach is leveraged on the recent new WSN capability of the reconstruction of upward routing paths [46–48], where individual upstream data packet paths from WSN nodes to the sink can be reconstructed at the sink with a minimal overhead of path encoding piggybacking to each data packet and updated in every data collection cycle. Our designed OSR protocol introduces opportunistic routing into the source routing, which is based on the parent set [49] of a node's upward routing, to exploit alternative downward paths to address wireless link dynamics. We devise a novel adaptive Bloom filter mechanism to efficiently encode and compress the source-route path. The probabilistic nature of the Bloom filter passively enables opportunistic routing for downward packet forwarding. In addition, when a downward link between a parent node and its child node is broken, active opportunistic routing is activated to find one or more other parent(s) in the child's parent set to continue the downward forwarding. OSR only requires that each node store its direct child set rather than its entire subgraph of descendants as in RPL (storing mode) or in ORPL (compressed entire subgraph) for making downward routing decision, and therefore, OSR is extremely scalable for constrained WSN sensor/actuator nodes. The proposed OSR is general and independent of the underlying link layer.

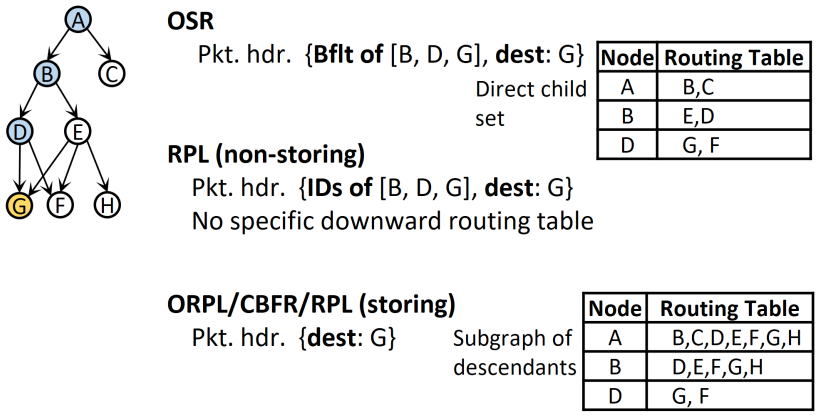


Figure 3.1. A conceptual illustration of downward packet delivery with OSR versus RPL/ORPL/CBFR.

To illustrate, Figure 3.1 shows an example of WSN downward actuation. To deliver a packet to node G, OSR includes the source-route encoded using Bloom filter and the destination in the packet header; each node only stores its direct children. In contrast, RPL (non-storing mode) specifies the raw source-route and destination in the packet header, whereas RPL (storing mode)/ORPL/CBFR only specifies the destination in the packet header, with each node storing the entire subgraph of its descendants either uncompressed in RPL or compressed using Bloom filter in ORPL/CBFR, thus suffering from scalability problem.

3.1 Related Works

Downward actuation protocols in WSNs can be classified into two categories: broadcast based and unicast/multicast based. A large portion of downward protocols, such as Drip [9], Glossy [50], and Opportunistic Flooding [51] are broadcast based that disseminate small data to the entire network (e.g., control packets). It would be very inefficient when using such broadcast based downstream protocols for individual node(s) actuation in LLNs, since the actuation commands would have to be flooded over the entire network. Recently, the demand to individual node(s) actuation arises

as heterogeneity becomes popular in WSN deployments, in which individual nodes play different roles in the network. RoCoCo [52] integrates the data collection and command dissemination. The command information is piggybacked in the routing beacons of the collection protocol, where the receivers' addresses are also included, hence enables dissemination to a subset of nodes. CBFR [26] and RBD [53] utilize the tree structure for downward routing. At each intermediate node, the route is determined by checking the whole subtree information stored at that node. Whereas RBD stores the raw addresses of the nodes in the subtree, CBFR utilizes counting Bloom filter to reduce the memory overhead and supports gradual forgetting for nodes mobility.

RPL [8] is a recent standard routing protocol for LLNs that supports upward, downward, and point-to-point (P2P) traffic patterns. RPL supports two modes for downward traffic, either through the entire subtree stored at each intermediate node's routing table (storing mode), or through the source-route specified at the root (non-storing mode). Storing mode suffers scalability problem with regard to the network size due to the limited memory in resource-constrained nodes, whereas non-storing mode suffers scalability problem with regard to the network diameter, due to the limitation in the frame size of the LLNs. A recent and comprehensive survey of RPL can be found in [16]. ORPL [25] brings opportunistic routing into RPL and improves the performance of RPL. Similar to CBFR, ORPL adopts Bloom filter/bitmap to represent node's subgraph (i.e., routing table) to reduce the memory overhead. ORPL using Bitmap only works for predefined static networks. When using ORPL on dynamic networks, the Bloom filter compression of a node's subgraph is propagated upward the collection tree in order for parents to update their routing tables. When the network is large, the Bloom filter size may also grow quickly and it would be inefficient for nodes to exchange their Bloom filters. Hence ORPL and CBFR also suffer from the scalability problem for large-size WSNs.

HB-DSR [54] is known to be the first source routing protocol that encodes the route into a Bloom filter. Bloom filters have been used in several approaches for multicast, such as [55] and [56]. However, those approaches are only targeted for wireline networks or mobile networks without significant resource constraints.

The IETF Roll Working Group is working on a draft Constrained-Cast [57] to consider using the Bloom filter to encode the source-routes in RPL (non-storing mode) for forwarding multicast traffic. However, this draft is still in the process and many details are unclear. Besides, this draft defines a few possible values of the Bloom filter size, as opposed to our adaptive Bloom filter size in OSR.

3.2 OSR Design

OSR introduces opportunistic routing into source routing, and creates an adaptive Bloom filter mechanism to encode the downward source-route. This section presents the core mechanisms of OSR including path representation, direct child set maintenance, and opportunistic routing.

3.2.1 Adaptive Bloom Filter for Path Encoding

In traditional source routing, the entire raw routing path is included in the packet header. As network grows, this approach consumes too much overhead or may even be infeasible for large-scale WSNs. For instance, containing a source-route of 20 hops using two-byte short address in IEEE 802.15.4 takes nearly one third of the maximum link layer frame size (i.e., 127 bytes). Thus, path encoding becomes a necessity for source routing to scale in resource-restricted WSNs. OSR exploits the Bloom filter [58, 59] to encode the source-route path, that is, a Bloom filter representing the source-route is included in the packet header instead of the raw path.

Bloom filter [58] is a space efficient probabilistic data structure that supports insertion and membership query. To insert an element into a Bloom filter of m bits, k independent hash functions are applied to deterministically generate k hash values

$h_i \in \{0, 1, ..., m-1\}$, and the corresponding bits are set to 1. For membership query, the element is hashed using the same set of hash functions. If all the k bits are matched in the Bloom filter, the element is considered being included/matched. A membership query may result in false positives, but never in false negatives. The false positive (FP) rate of a Bloom filter can be calculated according the following equation [58]:

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k, \tag{3.1}$$

where m is the length of the Bloom filter in bits, k is the number of hash functions, and n is the number of elements that are already encoded in the Bloom filter.

As an example of path encoding using Bloom filter, suppose in a large-scale WSN a path length $n = 20$, the length of the Bloom filter $m = 128$ bits, and $k = 3$ hash functions are used. The resulted probability of a false positive match (i.e., false positive rate) is 5.29%. Assuming each node address in the path occupies two bytes, using a 128-bit Bloom filter leads to 60% space saving compared to the raw path representation mechanism, indicating the effective use of Bloom filter in source routing in WSNs.

Since multi-hop WSNs usually need to be scalable in practice, using a fixed-length Bloom filter is inefficient. For instance, a too short Bloom filter would introduce high false positive rate for a long path, whereas a long Bloom filter may have more bits than a short raw path itself. We devise an adaptive path Bloom filter whose length m (bits) is proportional to the hop count H of the route:

$$m = \begin{cases} 8H & H \leq L \\ 8L & H > L \end{cases}, \tag{3.2}$$

where L is the maximum Bloom filter length of any encoded source route in bytes. Even with a minimum node ID (i.e., address) length of two bytes, the devised Bloom filter (i.e., Equation (3.2)) for path encoding leads to at least 50% space saving compared to the use of raw source-route in RPL (non-storing mode), indicating the poten-

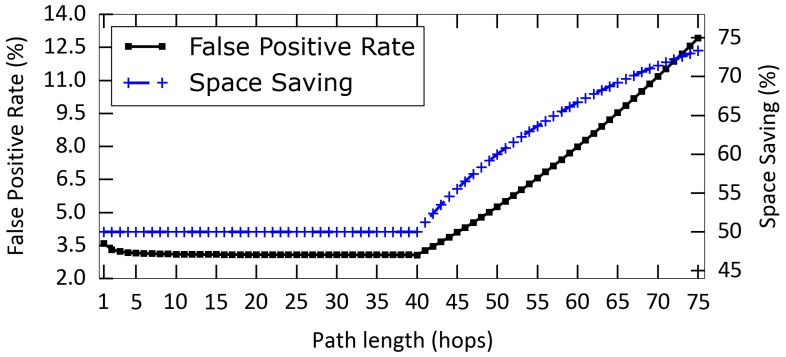


Figure 3.2. The false positive rate of adaptive Bloom filter and the corresponding space saving. Path length is H hops; Bloom filter length m is based on (3.2); the number of hash functions $k = 3$; $MAX\_BFLT\_LEN = 40$ bytes.

tial merit of our adaptive Bloom filter. Figure 3.2 demonstrates the analytical false positive rate based on (3.1) of the devised adaptive Bloom filter with $L = 40$ bytes and its corresponding space saving compared to the raw path. Clearly, our devised Bloom filter mechanism scales well with respect to the raw source route length. The resulting false positive rate is lower than 3.6% when hop counts do not exceed 40; the false positive rate actually drops as a path length increases. When a raw source route exceeds 70 hopes, our approach works fine only at a higher FP rate (<13%). In fact, our approach would still work for any long raw path potentially of hundreds (or even thousands) of hops at somewhat degraded performance (i.e., a higher FP rate). In contrast, traditional source routing, such as RPL non-storing mode, simply does not work for any raw path exceeding the maximum frame size of underlying link layer, which would be less than 64 hops for RPL with IPv6 address compression. If more false positives are tolerable, a shorter Bloom filter can be used to reduce packet overhead even more. In practice, the maximum Bloom filter length L (MAX_BFLT_LEN) is configurable for given WSN applications.

Due to the resource constraints in sensor/actuator nodes, hash functions employed in a Bloom filter should consume as less resource as possible. We adopt three hash

Algorithm 1: Bloom Filter Membership Query

Input: path Bloom filter $path\_bflt$, node's hash values h_i, $i \in \{1, ..., k\}$.

Output: $TRUE$ if the Bloom filter matches, $FALSE$ otherwise.

```
1 for i = 1 to k do
2     if (((1 ≪ h_i) ∧ path_bflt)==0) then
3         return FALSE
4     end
5 end
6 return TRUE
```

functions, namely Thomas Wang's hash function [60], Bob Jenkins' hash function [61], and FNV hash [62] in our Bloom filter structure due to their resource efficiency [26]. Note that, k's value can also be adaptive to the Bloom filter size and path length, as optimal $k = \frac{m}{n} \times ln2$ [58]. If optimal k is needed, we can further adopt SAX (Shift-and-Xor) hash to generate multiple hash values [25, 63].

When a downward packet is initialized at the sink, the source-route is encoded in a Bloom filter by ORing the Bloom filters of all the (intermediate) node addresses in the source-route. Upon reception of a downstream packet, a node checks if there exists any match of its direct child(ren) through the Bloom filter membership query of its each child, which can be done efficiently by an AND operation, as shown in Algorithm 1. If any of its direct child matches the Bloom filter, the packet is forwarded downward to the matched child node(s).

3.2.2 Direct Child Set

WSN nodes are usually highly resource limited. For example, a MicaZ node platform, only has 4K bytes of RAM. Even a TelosB node that is widely used in real-world WSN deployments only has 10K bytes of RAM. Existing approaches such as RPL [8] (storing mode), RBD [53], CBFR [26], and ORPL [25] require each node

to store/encode its entire subgraph of descendants for making downward routing decisions, causing the inherent scalability problem for large WSNs. In contrast, OSR only requires each node to store its one-hop direct children, referred to as the direct child set, for downward routing. Therefore, OSR is scalable on the size of the network with respect to node's memory.

As an illustration, we tested a data collection WSN application based on CTP in Indriya testbed densely deployed across three floors in a school building [64], which contained 95 available TelosB nodes at the experiment time. The test lasted for about 6 hours. We analyzed all the $(parent, child)$ pairs and computed the node distribution on the number of direct children they had. The statistics is shown in Figure 3.3. As we can see, around 50% of the nodes are leaf nodes, and no node has more than 12 direct children. In contrast, the subtree of an intermediate node can grow up to a size comparable to the entire network size, especially for the nodes near the sink.

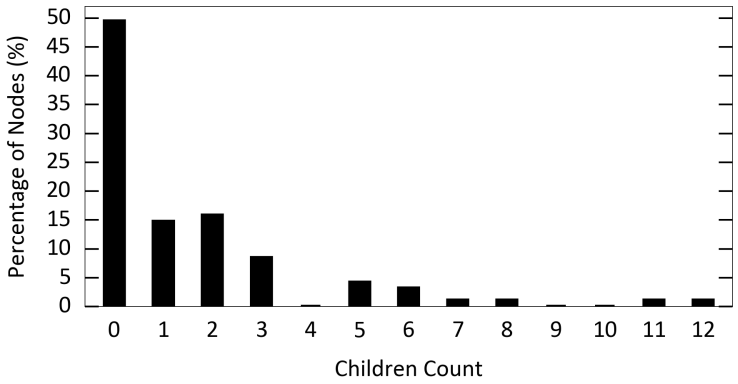


Figure 3.3. Node distribution on the number of direct children for the Indriya testbed with 95 available nodes.

OSR takes advantage of the underlying data collection routing protocol to establish the direct child set. When a node forwards an upstream packet, it inspects the packet header and adds the link layer sender's address to its direct child set. Some protocols provide easy access methods. For instance, CTP in TinyOS [36] offers an

Intercept interface for other applications to check the contents of a forwarded CTP packet.

To accommodate wireless link dynamics and hence the coming and leaving of a direct child, each direct child is associated with a time to live (TTL) flag. The TTL value decreases based on a periodic timer. When a child's TTL reaches 0, the child is removed from the direct child set. Every time a child is refreshed or added, its TTL value is reset. With TTL, the direct child set should be able to cover all the $(parent, child)$ relationships in the data collection network within the time window of length TTL. If an intermediate node is included in the downward path Bloom filter but none of its direct children is, it is highly possible that the node itself is a false positive.

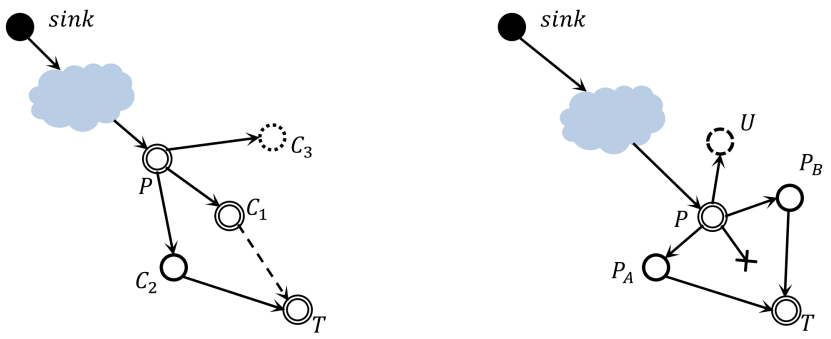
3.2.3 Opportunistic Routing

In data collection WSNs (e.g., the ASWP WSN testbed described in Chapter 4), a node may have multiple candidate parents that are able to forward its data packets within a time window, forming a parent set [49] of the node. Moreover, the parent nodes belonging to a same parent set have a high probability being within the transmission range of each other. Based on these observations, OSR introduces opportunistic routing into the traditional source routing by exploring alternative routes based on node's parent set to improve the reliability of downward routing in dynamic WSNs. Note that in OSR, nodes are not aware of their parent set explicitly. Instead, a node implicitly joins a child's parent set when it adds the child into its direct child set.

The introduced OSR opportunistic routing acts in two aspects. First, the probabilistic nature of the Bloom filter of the source-route would be able to potentially, albeit in a passive way, explore the parent nodes of an in-route node not given in the source-route but opportunistically matched by false positives. This in fact provides alternative route(s), beyond the given source-route, for downward packet forwarding.

During the OSR downward routing process, whenever a node has multiple matched children in the path Bloom filter, OSR transmits the packet to all the matched children by local multicast. In the case that the matched child(ren) due to the false positive(s) is/are also in the parent set of the grandchild in the downward path, the packet is then opportunistically delivered to the grandchild. Therefore OSR, to some extent, turns false positives in the Bloom filter into potential opportunities for downward packet forwarding, which can improve the reliability without any false positive recovery scheme. Second, OSR actively performs opportunistic routing by requesting the other parent nodes in the parent set of a child node to assist packet forwarding whenever a normal downstream unicast based on the source-route fails. Due to the drastic wireless link dynamics in low-power WSNs, a source-route may be obsolete when the downstream packet arrives at an intermediate node. Source routing fails if the next hop in the source-route becomes unreachable at an intermediate node. In such an event, the intermediate node will broadcast the packet to its neighborhood, hoping that one or more of its neighbors belonging to the parent set of the next-hop child node opportunistically receive(s) it. Upon reception of a broadcast packet, the node will check whether any of its direct child is in the source-route. If yes, which indicates the node likely belongs to the next-hop child's parent set, the node would forward the packet to the matched child(ren).

Figure 3.4 illustrates how opportunistic routing is conducted in OSR. The passive opportunistic routing is shown in Figure 3.4(a). The source-route specifies $[\cdots P \rightarrow C_1 \rightarrow T]$. Nodes C_1, C_2, and C_3 are children of node P which are matched in the path Bloom filter of the downward packet. In addition to $(C_1 \rightarrow T)$, node C_2 is also in the parent set of grandchild node T, hence $(C_2 \rightarrow T)$ is an alternative path explored through the passive opportunistic routing. The active opportunistic routing is illustrated in Figure 3.4(b). Node T is a child of node P as specified in the downstream source-route. When P fails to deliver the packet to T, it broadcasts the packet to its neighbors. Three neighbors have received the broadcast. Whereas neighbor U is not in the parent set of T and will ignore the packet, neighbor P_A



(a) Passive opp. routing. (b) Active opp. routing.

Figure 3.4. Illustration examples of (a) passive opportunistic routing and (b) active opportunistic routing in OSR. The source-route is marked as circles with double line border.

and P_B will forward the packet to T because they are in the parent set of T. Thus, the obsolete link from P to T is successfully bypassed by the opportunistic routing activated by node P.

3.2.4 Downward Routing Decision

Unicast is the basic MAC layer transmission scheme used in OSR to deliver a downstream actuation packet. If any unicast fails after its maximum retransmissions, broadcast is used for active opportunistic routing. In addition, if a node has multiple direct children that are included in the path Bloom filter, it uses local multicast to deliver the packet to all the matched children for passive opportunistic routing.

OSR includes two-bit information in a downstream packet header to distinguish the three transmissions types (i.e., unicast, local multicast, or broadcast) of the packet. A received packet is processed based on the two-bit header information accordingly. While a multicast reception would require each receiving node to check its membership to the received path Bloom filter, broadcast does not require each receiving node to check its membership. In the case of lacking the support of multicast

in the MAC layer, multicast can be implemented by broadcast. OSR would benefit from multicast-supported MAC layers. Unicast reception indicates the receiver node must be included in the path Bloom filter without the check of the membership.

Retransmissions are used for all the transmission types to ensure link layer reliability. Unicast utilizes link layer one-to-one acknowledgment which is available in most of the radio stack implementations. If the packet is not acknowledged by the child, it is retransmitted, until the maximum number of retransmissions is reached. Multicast can either use active acknowledgment or passive acknowledgment. Note that in the case that multicast has not been implemented in the link layer (e.g., BOX-MAC [35] in TinyOS [36]), the acknowledgments would have to be handled by OSR directly. Due to the nature of wireless communication, passive acknowledgment, by overhearing the children nodes' transmissions of the same packet, is more energy efficient. The multicast is conducted repeatedly for a maximum number of times (N_1) until all the acknowledgments are overheard. Broadcast does not need acknowledgment so it is always transmitted for N_2 times. The values of N_1 and N_2 should be large enough to ensure at least one successful link layer delivery.

We have devised the OSR algorithm (i.e., Algorithm 2) for the routing decision making process at an intermediate node. If a node receives a multicast packet and passes the membership check, it starts to check its direct child(ren); otherwise the packet is ignored. If a node receives a unicast or broadcast packet, it immediately starts to check the membership of its direct children in the path Bloom filter (*path_bflt*). If a node has multiple children included in the *path_bflt*, the packet is forwarded using local multicast. If there is only one matched child, the packet is forwarded by unicast. Any unicast failure would trigger active opportunistic routing through broadcasting. If there is no any matched child, the packet is ignored, since it is of high probability that the node itself is a false positive. In our design, each node keeps a history of recently received downstream packets to avoid duplicates and forwarding loops. Duplicate packets are ignored immediately. A time-to-live (TTL)

Algorithm 2: Downward OSR

Notations:

path_bflt: the Bloom filter contains the IDs along the downstream path.

matched_count: the number of matched children in *path_bflt*.

tx_typet: the transmission type of the recieved downward packet.

```
1  begin
2      if packet is duplicate then
3          return
4      end
5      if tx_type is Multicast then
6          if local ID is NOT included in path_bflt then
7              Ignore the packet and return
8          end
9      end
10     Check children for match
11     if (matched_count > 1) then
12         Multicast the packet
13     else if (matched_count > 0) then
14         Unicast the packet
15         if unicast fails then
16             if tx_type is not Broadcast then
17                 Broadcast the packet /*Opportunistic routing*/
18             end
19         end
20     else
21         /*no matched children, ignore the packet*/
22     end
23 end
```

field (e.g., initialized as two times of the path length) is also associated with the packet to avoid infinite forwarding loops.

3.2.5 OSR Optimizations

Previous sections have described the basic mechanisms of OSR. In this section, we present a few directions to optimize the performance of OSR under different network conditions.

First, the direct child set can be expanded using snooped children if the topology is relatively stable. For tree-like collection protocols (e.g., CTP), a node usually selects the parent with the best path to the sink, and not to switch parent unless there is a significant change in link conditions. However, less significant changes in link conditions can also cause delivery failures and add delay due to the time between the link broken and the fix of the topology by the upward routing protocol. When a nodes parent set size is only 1, opportunistic routing in OSR would not work for this node since there is no another candidate forwarder. To facilitate opportunistic routing in OSR, a node can actively join an indirect child's parent set by overhearing the indirect child's transmission. Tree-like upstream routing protocols are usually cost-based. A node identifies an indirect child if it detects an overheard upstream packet has a higher routing metric compared to its own, for example, based on the following equation:

$$rcvd\_pkt\_metric > local\_metric + \omega, \tag{3.3}$$

where $rcvd\_pkt\_metri$ is the upward routing metric of the overheard data collection packet and $local\_metric$ is the node's own upward routing metric. ω is a control parameter indicating how aggressive a node can add an indirect child, similar to that of ORW and ORPL for adding forwarders.

Second, adapt the number of maximum multicasts/broadcasts to link quality. OSR uses multicast to deliver the packet when multiple children are matched in the path Bloom filter at an intermediate node. Even if passive acknowledgment is used

in multicast, a node may not be able to overhear its childrens transmissions due to interferences or noises. Hence, a node may use up all the multicast retransmissions. The adaptation can reduce the transmission overhead of OSR. Instead of a predefined fixed maximum number of retries, a node can adapt the retries of multicast/broadcast according to the current link qualities to the neighbors, as long as the successful link layer delivery is ensured. As an example, the maximum number of multicast retries can be computed as follows:

$$max\_retries = 1 + \frac{1}{|S_{ch}|} \sum_{c \in S_{ch}} \frac{1}{lprr_c}, \tag{3.4}$$

where $lprr_c$ is the link layer packet reception rate of a node to its child c in the direct child set S_{ch}. The second term is the expected number of transmissions necessary for the successful packet delivery averaged on all the children, with one additional transmission (i.e., the first item in (3.4)) to ensure reliability to all children. Since OSR does not transmit beacons, the link layer packet reception rate is estimated through the unicast from the node to its children. Hence there is a relatively long initialization phrase. If the routing table of the underlying collection protocol is available, then the estimation can be computed immediately based on the link estimations. More sophisticated calculation approaches can also be applied.

The maximum number of broadcast retries can be computed in similar manner if the neighbors' information is available.

3.3 Analytical Scalability

This section presents the analytical scalability of the protocols RPL, ORPL, and OSR. For RPL storing mode and ORPL, the scalability is defined with regard to the memory occupation to store the subtree information rooted at each node. For RPL non-storing mode and OSR, the scalability is defined with regard to the packet overhead required to store the source-route information. We assume the network is uniformly distributed of size N. Other notations are listed in Table 3.1.

Table 3.1.
Notations for Analytical Scalability

Parameter	**Definition**
m	The length of the Bloom filter in bits.
k	The number of hash functions.
n_t	The size of the subtree rooted at the node.
n_l	The length of the source-route.
N	The size of the network.
N_{nb}	The size of the neighborhood of the node.
N_{ch}	The size of the direct child set of the nodes in the neigborhood.
C	The number of true children among the neighborhood.
P	The upper bound of the false positive rate to avoid packet storm problem.

3.3.1 Scalability of RPL

For both RPL modes, the scalability is straightforward.

- RPL storing mode: the memory required to store the subtree of a node is in the order of $O(n_t)$ bytes, where n_t is the size of the subtree rooted at the node. Specifically, if each node address takes 16 byte (e.g., IPv6 address), it would be in the order of $O(16n_t)$ bytes. For the nodes near the root, the memory would be in the order of $O(N)$ bytes since the subtree size is comparable to the size of the network.

- RPL non-storing mode: the number of bytes required to store the source-route is proportional to the number of addresses in the source-route, which is $O(n_l)$. If each node address takes 16 bytes, it would be $O(16n_l)$ bytes; if each node address is 2 bytes, it would be $O(2n_l)$ bytes. To reach the farthest nodes of the

network, the path length n_l is in the order of $O(\sqrt{N})$. Thus, the number of bytes for storing the source-route in the packet is $O(\sqrt{N})$.

3.3.2 Scalability of ORPL and OSR

ORPL and OSR both use Bloom filter based forwarding. Our goal of the scalability analysis is to find the lower bound of the Bloom filter size m which avoids the *packet storm problem* [55]. The packet storm problem occurs if a false positive will cause more than one false positive in the next forwarding node on average. Theoretically, if the packet is broadcasted, the packet storm problem will occur if the product of the node degree and the false positive rate exceeds one, $(d-1)\cdot p \geq 1$, where d is the node degree and p is the false positive rate. Note that, excluding the node from which the packet is received, there are $d-1$ potential neighbors at each forwarding node.

Recall that the false positive rate of a Bloom filter is

$$p = \left(1 - \left(1 - \tfrac{1}{m}\right)^{kn}\right)^k,$$

where n is the number of elements in the bloom filter, m is the length of the bloom filter in bits, and k is the number of hash functions.

To avoid the packet storm problem, the purpose is to ensure

$$p < P, \tag{3.5}$$

for some threshold probability P.

Substitute (3.1) to (3.5), we get

$$\left(1 - \left(1 - \tfrac{1}{m}\right)^{kn}\right)^k < P.$$

It is easy to obtain the lower bound of m:

$$m > \frac{1}{1 - \left(1 - P^{\frac{1}{k}}\right)^{\frac{1}{kn}}}. \tag{3.6}$$

Since $P^{\frac{1}{k}} < 1$ and $\frac{1}{kn} \cdot P^{\frac{1}{k}} << 1$ (as kn is usually larger than 10), then (3.6) becomes

$$m > \frac{1}{1 - \left(1 - P^{\frac{1}{k}}\right)^{\frac{1}{kn}}} \approx \frac{1}{1 - \left(1 - \frac{1}{kn} \cdot P^{\frac{1}{k}}\right)} = kn \cdot P^{-\frac{1}{k}}. \tag{3.7}$$

ORPL

For ORPL, each packet is transmitted through anycast, that is, the packet is broadcasted then acknowledged by the first node whose subtree includes the target node. So all the neighborhood would check their Bloom filter upon overhearing of the packet.

Suppose the size of the neighborhood is N_{nb}, and for simplicity, suppose the subtree for each neighbor is of the same size n_t. To avoid the packet storm problem, we have

$$p \cdot (N_{nb} - 1) < 1,$$
$$p < \frac{1}{N_{nb}-1}.$$

Based on (3.5), we obtain

$$P \geq \frac{1}{N_{nb} - 1}. \tag{3.8}$$

Substitute (3.8) to (3.5) we obtain the final expression of m for ORPL:

$$m > kn_t \cdot P^{-\frac{1}{k}} \geq kn_t \cdot (\frac{1}{N_{nb} - 1})^{-\frac{1}{k}} = kn_t \cdot (N_{nb} - 1)^{\frac{1}{k}} \tag{3.9}$$

Note that n_t is in the order of $O(N)$, and N_{nb} is related to the density of the network.

For ORPL to be more scalable than RPL storing mode, the coefficient in (3.9) must be smaller than that of RPL storing mode. For instance, if each node address takes 2 bytes (16 bits), then the coefficient of ORPL must sastify

$$k \cdot (N_{nb} - 1)^{\frac{1}{k}} < 16,$$
$$N_{nb} < \frac{16}{k}^{k} + 1, \tag{3.10}$$

where k is the number of hash functions, which is usually an integer smaller than 10. The maximum range of N_{nb} with varies k values is shown in Figure 3.5. As we can

see, if the neighborhood size is smaller than the upper bound defined by the curved line, ORPL would always consume less memory then RPL storing mode.

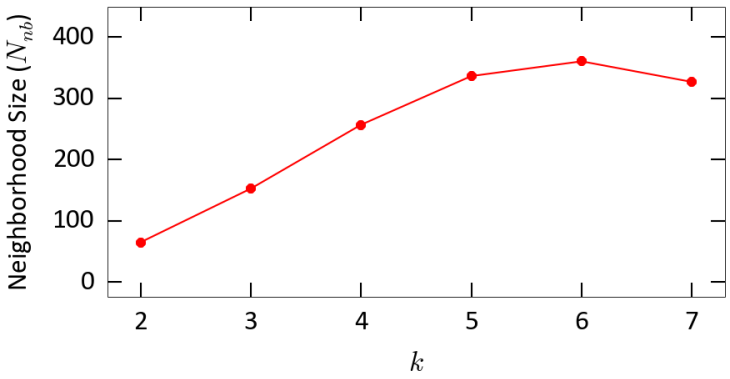


Figure 3.5. Upper bound of the neighborhood size (N_{nb}) with varies number of hash functions k, given the coefficient upper bound of 16 bits.

OSR

In OSR, if a packet is transmitted using unicast, then there is no packet storm problem. If a packet is transmitted using broadcast, then the next round of transmission would only be unicast or multicast. Only the packets transmitted using multicast can probably cause the packet storm problem.

Suppose the size of the neighborhood is N_{nb}, and for simplicity each neighbor has the same number of N_{ch} children in its direct child set. When a node receives a multicast packet, it has to pass the membership check of the path Bloom filter before checking its children. Here we assume C of the neighbors are true children that included in the path Bloom filter. Thus, there would be $(N_{nb} - C - 1) \cdot p$ false positive neighbors that pass the membership check.

For each neighbor that passes the membership check, it would forward the packet when it finds a match in its direct child set. The probability for a neighbor to have at least one false positive matched child in its direct child set is

$$p_m = 1 - (1-p)^{N_{ch}}, \tag{3.11}$$

where $(1-p)$ is the probability that a node is not a false positive, and $(1-p)^{N_{ch}}$ is the probability that none of the N_{ch} children are false positive.

To avoid the packet storm problem, we need to ensure

$$p_m \cdot (N_{nb} - C - 1) \cdot p < 1. \tag{3.12}$$

Substitute with (3.11) we have

$$\begin{aligned} (1-(1-p)^{N_{ch}}) \cdot (N_{nb} - C - 1) \cdot p &< 1, \\ p(1-(1-p)^{N_{ch}}) &< \frac{1}{N_{nb} - C - 1}. \end{aligned} \tag{3.13}$$

Usually $p << 1$, we can expand the term $1-(1-p)^{N_{ch}}$ by the first two orders and obtain

$$\begin{aligned} p(1-(1-p)^{N_{ch}}) &= p \cdot \left(1 - \left(1 - N_{ch} \cdot p + \tfrac{1}{2} N_{ch}(N_{ch}-1)p^2\right)\right) \\ &= p^2 \cdot N_{ch} \cdot (1 - \tfrac{1}{2} p(N_{ch}-1)) \end{aligned}$$

Substitute to (3.13) we obtain

$$p^2 \cdot (1 - \frac{1}{2} p(N_{ch}-1)) < \frac{1}{N_{ch} \cdot (N_{nb} - C - 1)} \tag{3.14}$$

Since usually $p << 1$ and N_{ch} is usually in the order less than ten, then $p(N_{ch}-1) \leq 1$ and leads to $\frac{1}{2} p(N_{ch}-1) \leq \frac{1}{2}$. Thus,

$$\begin{aligned} p^2 \cdot \frac{1}{2} \leq p^2 \cdot (1 - \frac{1}{2} p(N_{ch}-1)) &< \frac{1}{N_{ch} \cdot (N_{nb} - C - 1)} \\ p &< \left(\frac{2}{N_{ch} \cdot (N_{nb} - C - 1)}\right)^{\frac{1}{2}} \end{aligned}$$

Substitute to (3.5) we obtain

$$P \geq \left(\frac{2}{N_{ch} \cdot (N_{nb} - C - 1)}\right)^{\frac{1}{2}}. \tag{3.15}$$

And finally

$$\begin{aligned} m > kn_l \cdot P^{-\frac{1}{k}} &\geq kn_l \cdot \Big(\Big(\frac{2}{N_{ch} \cdot (N_{nb} - C - 1)} \Big)^{\frac{1}{2}} \Big)^{-\frac{1}{k}} \\ &= kn_l \cdot \Big(\frac{N_{ch} \cdot (N_{nb} - C - 1)}{2} \Big)^{\frac{1}{2k}}, \end{aligned} \tag{3.16}$$

where n_l is the number of nodes in the source-route, which is in the order of $O(\sqrt{N})$, the same as RPL non-storing mode. N_{ch} and N_{nb} are related to the density of the network. N_{ch} also relates to the extent of routing dynamics.

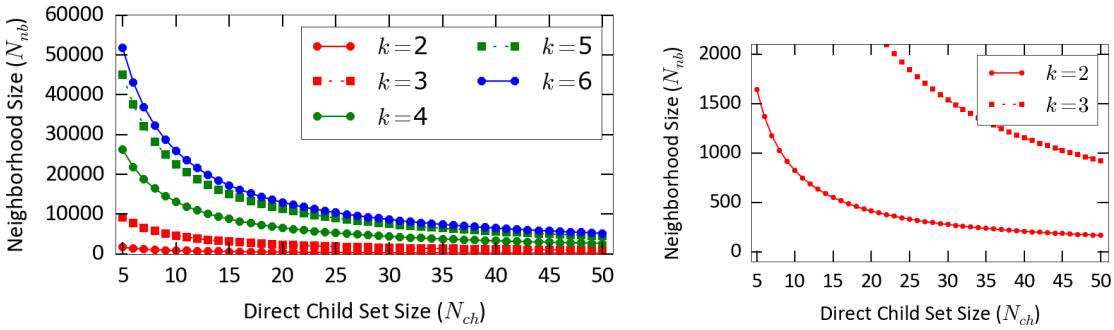
For OSR to be more scalable than RPL non-storing mode, the coefficient in (3.16) should be smaller than that of RPL. For instance, if each node address takes 2 bytes (16 bits), then the coefficient of OSR must satisfy

$$k \cdot \Big(\tfrac{N_{ch} \cdot (N_{nb} - C - 1)}{2} \Big)^{\frac{1}{2k}} < 16.$$

Usually the number of matched children $C \geq 2$ for OSR multicast transmission, then we have

$$N_{ch}(N_{nb} - C - 1) \leq N_{ch}(N_{nb} - 3) < 2\Big(\frac{16}{k}\Big)^{2k}. \tag{3.17}$$

Figure 3.6 illustrates the upper bound of N_{ch} and N_{nb} with varies number of functions k given the coefficient upper bound to be 2 bytes (16 bits). As we can see, the available range of N_{nb} and N_{ch} is quite large for the given upper bound. In fact, the coefficient of (3.16) is usually less than 1 byte for many WSN deployments. For instance, suppose $N_{nb} = 25$, $k = 3$, $N_{ch} = 10$, and $C = 2$, then $m > 6.67n_l = 0.83n_l$ bytes. The result indicates that OSR usually requires less than 1 byte to store a node in the source route compared to 2$\sim$16 bytes (depending on the address compression) per node in of that in RPL non-storing mode. Our newly designed adaptive Bloom filter (3.2) fits well in this range, demonstrating its scalability.



(a) Varies k.

(b) $k = 2, 3$.

Figure 3.6. Upper bound of the neighborhood size (N_{nb}) for varies direct child set sizes (N_{ch}) given the coefficient upper bound being 2 bytes (16 bits). (a) varies number of functions k; (b) $k = 2, 3$.

4 OSR IMPLEMENTATION AND EVALUATION

This chapter presents our implementation of OSR and the evaluation through a series of simulations and real-world WSN testbed experiments in comparison against existing protocols. We plan to make our OSR implementation, including all the test applications, publicly available.

4.1 OSR Implementation

OSR takes advantage of the data collection protocol to build the direct child set and find the reverse path to each individual node, resulting in negligible overhead for routing maintenance. Our OSR is implemented using nesC based on TinyOS 2.1.2, working in concert with CTP or CTP+EER [49] as the underlying data collection protocol.

To begin with, we define the OSR packet structure as shown in Figure 4.1. The *seqno* is the sequence number of the OSR packet, which is only increased at the sink node when an OSR packet is issued. It is a 16-bit integer that is large enough to reduce the probability of overflow, especially for large networks. The *rnd* is a random integer, which is generated by the sink at the same time as the sequence number is increased when an OSR packet is issued. In OSR, the tuple (*seqno*, *rnd*) uniquely identifies a downward packet and is used to detect duplicate packets at the nodes. The *TX* is a two-bit field indicating the transmission type of the received OSR packet, as described in Section 3.2.4. There are several *reserved* bits that can be used in the future. The *path_len* is the length of the source-route, which can be used to decide the length of the path Bloom filter according to (3.2). The *ttl* field indicates the remaining lifetime of the packet, when it reaches 0 the packet is dropped. The *target_id* is the address of the target node for actuation. In TinyOS, each node address is 2 bytes.

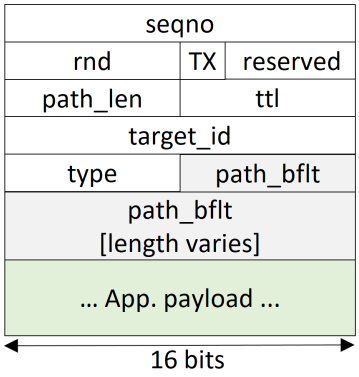


Figure 4.1. OSR packet format.

Each *type* identifies a specific payload structure that carried in an OSR packet. The *path_bflt* is the Bloom filter that encodes the source-route. Its length varies based on the length of the source-route.

The full functionalities of OSR are divided into the sink logic and the node logic, and are implemented in three major components:

- *Path Encoding.* Only at the sink side. To issue an OSR packet, the source-route to reach the target node must be already known (as required by source routing). Usually a computer gateway is used to issue downward packets on demand. In such situation, the raw source-route is provided for the sink node by the gateway, then the sink node would encode the source-route into the path Bloom filter.

- *Direct Child Set.* At both the node side and the sink side. OSR only requires a single interface *Intercept* provided by CTP to build its direct child set. When CTP receives an upward packet to forward, OSR uses the *Intercept* interface to inspect the link layer header of the packet, and adds the link layer sender (i.e., its direct child) to the direct child set.

- *Packet Forwarder.* On the sink side, the sink node checks its direct child set for matches in the path Bloom filter and transmits the packet based on the number

of matched children. On the node side, when an OSR packet is received, the packet is processed based on Algorithm 2.

As described in Section 3.2.5, if overhearing is used to expand the routing table of OSR, the *CtpInfo* and *CtpPacket* interfaces of CTP are used in order to retrieve and compare the upward routing metric of the overheard children and the receiving node. If the routing metric of a overheard child satisfies (3.3), the child is added to the snooped children table, and will be checked for Bloom filter membership when an OSR packet is received. The structure of OSR implementation is illustrated in Figure 4.2.

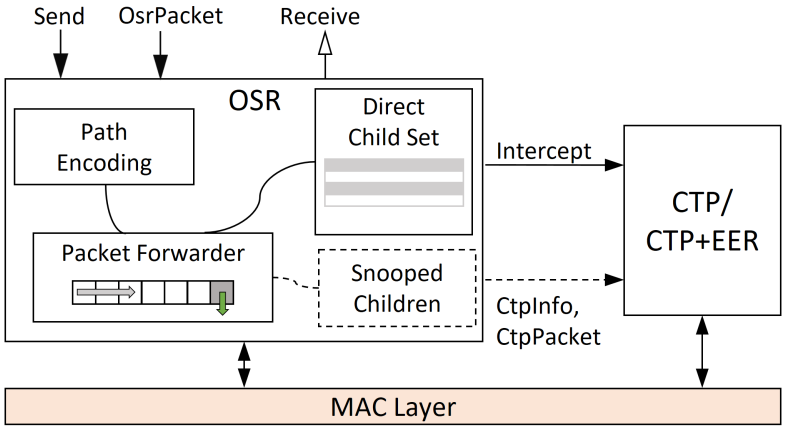


Figure 4.2. OSR implementation with CTP/CTP+EER. *Path Encoding*, *Send*, and *OsrPacket* are for the sink side only. *Receive* is for the node side only.

Our current OSR implementation basically provides three interfaces for the upper layer. Before calling *Send*, the *OsrPacket* interface is used to configure some fields in the OSR packet header, such as path length and target address. These two interfaces are only used at the sink node. On the node side, when an OSR packet reaches the target node, the *Receive* interface is used to signal the upper layer.

4.2 Evaluation Methodology and Setup

OSR is evaluated against RPL (both storing and non-storing modes), ORPL[1], and Drip. We consider both ContikiRPL [65] and TinyRPL [66], the two most widely used open-source RPL implementations. ContikiRPL supports both storing mode and non-storing mode, whereas TinyRPL only supports storing mode. ORPL is implemented based on ContikiRPL storing mode [25]. The test application of RPL is written based on the examples that come with the RPL and ORPL implementations.

We have evaluated OSR through simulators Cooja and TOSSIM, and the publicly available WSN testbed Indriya.

For all the experiments, the following configurations were used unless stated otherwise. OSR which was implemented based on Algorithm 2 is referred to as the basic version. The size of direct child set was set to 20 with the child *TTL* value initialized to 4, which was updated every collection cycle; the maximum number of downward unicast retransmissions was 10, and the maximum number of multicast/broadcast retransmissions was 5; the maximum Bloom filter length *MAX_BFLT_LEN* was set to 16 bytes for low false positive rate. Note that that the actual length of the Bloom filter (in bytes) carried in a packet equals to the hops of the downstream path if it is smaller than *MAX_BFLT_LEN*. Our current OSR implementation increases 674 bytes in RAM usage and 3813 bytes in ROM usage, respectively, on top of the CTP (with a simple application) which occupies 1720 bytes in RAM and 17904 bytes in ROM, respectively, for TelosB nodes.

We consider the following key performance metrics. First, scalability, indicated by the protocol's performance as downward path length increases. Second, the network downward Packet Delivery Ratio (PDR), defined as the ratio between the number of actuation packets received by the target nodes and the total number of packets sent by the sink. Third, the Duty Cycle (DC), the portion of time when the radio is on in low power MAC, as the measurement of energy efficiency with the same implementation

[1]https://github.com/simonduq/orpl

platform. Then, transmission Cost (TxCost) is defined as the average number of transmissions of the network to deliver a downstream packet, an indirect indicator of the energy efficiency. We also show the collection protocol's packet delivery ratio (cPDR) as an indicator of the network condition when the test was conducted. For all the performance results the sink's transmissions are not included, as the sink/root is considered to have unlimited power supply.

4.3 Evaluation in Cooja

We first conducted simulations in Cooja [67] using the TelosB platform to evaluate the scalability of OSR against RPL and ORPL.

In the smart city scenario, urban structures may shape the network to a peculiar topology [21]. Inspired by [21], we evaluate the scalability limit of the protocols in a quasi linear network topology with small twigs, which may be quite common in urban areas. The quasi linear network consists of 74 nodes and builds up to 68 hops, with the sink/root being at one end (as illustrated in Figure 4.3). We use the Unit Disk Graph Medium (UDGM) with exponential distance loss as radio model and a maximum link quality of 90% to account for uniform random noise. A node sends upward data packets randomly with an average interval of 10 minutes. After 20 minutes of network initialization, the sink starts to send an actuation packet every 10 seconds to a randomly selected target node. Upward packet payload is 60 bytes and downward packet payload is 20 bytes. Table 4.1 lists the compiled RAM usage of the test application with each protocol under different network size configurations. OSR was configured with the same MTU as the default in TinyRPL (i.e., 112 bytes). Since RPL storing mode has scalability issues in terms of memory when the routing table size increases, the routing table size in ContikiRPL (storing) and TinyRPL is configured to be 50, which results in 9104 bytes and 8160 bytes of memory footprint for ContikiRPL (storing) and TinyRPL, respectively. Each simulation ran for 4 hours, in which a total number of 1320 downward packets were sent out.

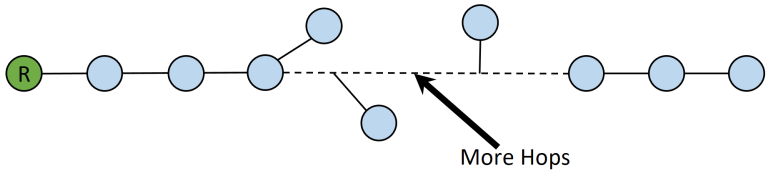


Figure 4.3. The illustration of the quasi linear topology. Root is at the left end.

Table 4.1.
Comparison of RAM Sizes for TelosB Platform

WSN Size	**RAM Usage (bytes)**				
	TinyRPL	**ContikiRPL (S)**	**ContikiRPL (NS)**	**ORPL**	**OSR & CTP**
2	5952	7280	7164	9710[a]	3958
50	8160	9104			
74	9264	10016			
225	+5972[b]	+5516			
400	+14022	+12164			

[a]ORPL includes a whole set of tools for logging.
[b]Numbers with "+" indicates the amount that overflowed the TelosB RAM space.

The evaluation results are shown in Table 4.2. OSR has successfully reached all the nodes (i.e., up to 68 hops) along the linear topology with 99.86% downward PDR. In contrast, all RPL implementations suffer scalability problems. ContikiRPL (non-storing) has only reached as far as 32 hops from the sink, far less than the theoretical threshold of 64 hops. Consequently, ContikiRPL (non-storing) has a poor PDR since more than half of the nodes are unreachable due to its scalability issue. On the other hand, the maximum reachable hop count in both ContikiRPL and TinyRPL storing modes as well as in ORPL is ad hoc, depending on the dynamics of nodes'

Table 4.2.
Scalability Comparison on Quasi Linear Network

Protocol	Max Reachable Hops	PDR(%)
OSR	68	99.86
ContikiRPL (NS)	32	33.31
ContikiRPL (S)	–	76.91
TinyRPL	–	27.95
ORPL	–	63.06

limited routing table establishment. As we can see, their PDR performances are also significantly lower than that of OSR.

To better understand the protocols' scalability, we show in Figure 4.4 the network PDR up to the first 25 hops in the downward routing. As we can see, the PDR of TinyRPL drops quickly as the path length increases. ContikiRPL (non-storing) experiences a steep drop after the 17th hop, where the downward packet fragmentation begins due to the long downward path length. ContikiRPL (storing) maintains high PDR for most of the time, but suffers from performance drop at several random hops, which is likely due to the probabilistic occupation in their ancestor nodes' routing tables at times, which is also observed for TinyRPL. In contrast, OSR achieved nearly 100% PDR regardless of the downward path length. ORPL also achieved high PDR within the first 25 hops.

To summarize, ContikiRPL (non-storing) suffers scalability problem regarding the network diameter, whereas ContikiRPL (storing) and TinyRPL suffers scalability problem regarding the network size. IP fragmentation harms the performance of ContikiRPL (non-storing) significantly. ORPL also severely suffers from the scalability. We speculate that the network linear topology might have affected the anycast mechanism of ORPL. In contrast, OSR scales significantly better than all of the RPL implementations and ORPL. In fact, since OSR uses localized direct child set, it does

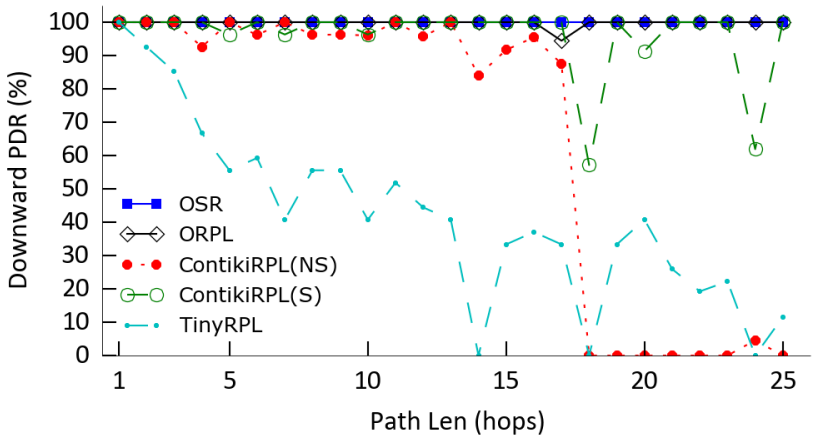


Figure 4.4. Downward PDR for linear topology based on downward path length in the first 25 hops.

not suffer as network size increases. Moreover, due to its Bloom filter based path encoding, OSR should be able to work with any path length of hundreds of hops.

4.4 Evaluation in Indriya

Next we evaluate the reliability and energy efficiency of OSR in comparison with TinyRPL and Drip in the Indriya testbed. ContikiRPL and ORPL were not included since they are based on the Contiki MAC on the Contiki platform which is very different from the TinyOS platform. As we know, energy efficiency is heavily dependent on the platform in addition to routing protocol[2].

The Indriya testbed consisted of 95 TelosB nodes during the experiment time. The testbed was configured to be low power for our experiments. Node 31 at the corner on the first floor was selected as the sink to maximize the network diameter. The *MAX_BFLT_LEN* was configured to 16 bytes.

[2]However, we have conducted simulations in Cooja to compare the relative energy efficiency of the protocols with their collection-only baseline. The test application is similar to that of Section 4.3, with a random topology. Compared to ContikiRPL baseline, the storing-mode has increased the node average duty cycle for 13.80%, non-storing mode has increased the average node duty cycle for 10.34%. Compared to a CTP baseline (in TinyOS), OSR has increased the average node duty cycle for 12.59%.

Table 4.3.
Comparison between OSR and TinyRPL on Low Power Indriya Testbed with 47 Nodes

	PDR (%)	**Duty Cycle (%)**
CTP Baseline	–	4.26 ± 0.07
TinyRPL	89.45 ± 0.04	18.67 ± 0.45
OSR	97.80 ± 0.01	4.43 ± 0.17

4.4.1 Comparison against RPL

Since TinyRPL could not work on the entire Indriya testbed, we conducted several experiment trials (30 minutes each) only using a half size of the testbed (i.e., 47 nodes with odd IDs) to evaluate OSR versus RPL. The test application collected data packets for the first 10 minutes in each trial for the network's initialization. The sink then sent downward packets to a randomly selected individual node every 10 seconds. Nodes were configured to be low power with a sleep interval of 1 seconds using the default TinyOS MAC (i.e., BoX-MAC [35]), whereas the sink was configured to be always on. We also conducted a pure CTP application as the baseline, where sink sends no downward packets, and node stops sending upward packets after the network's initialization.

Table 4.3 shows the performance results of OSR versus TinyRPL averaged on four trials. As we can see, OSR (with CTP) performs significantly better than TinyRPL on both the downward PDR and the duty cycle. TinyRPL's high duty cycle is mainly caused by its high DAO packet rate. We believe a careful tuning of the DAO rate could benefit TinyRPL's performance, however, it requires a systematic adjustment and is not the focus of this work. In particular, OSR itself only adds a very little to the duty cycle compared with the CTP baseline, demonstrating its energy efficiency.

4.4.2 Comparison against Drip

Next, we compare OSR versus Drip in Indriya with all 95 available nodes. Drip is built on top of Trickle [43] for dissemination of the entire network. For unicast actuation, a *target_id* field is included in Drip's application packet to ensure only the targeted node would act when the command is received.

The WSN test application has a collection cycle interval of 4 minutes. Nodes operated on low power with a sleep interval of 1 second. The sink was configured to be always on. The sink started to issue one downstream packet to a randomly selected node every minute starting from the beginning of the third cycle if the node's upward packet was collected. Each experiment ran for about 6.8 hours, with 400 downward actuation packets were issued.

In order to evaluate the effects to the network energy efficiency of OSR and Drip, we conducted a baseline experiment which ran a pure data collection application with CTP using the same configuration as Drip. The data collection application was tested twice and the results were averaged to smoothen out the effect of network dynamics. The results of the baseline tests showed an average collection packet delivery rate cPDR of 93.47% and an average duty cycle of 3.32%.

Overall Result

The variances in Indriya's physical environment (e.g., human activities) caused fluctuations in protocol performances since the experiments were conducted at different times. The collection packet delivery rate cPDR and nodes distribution on their average path length from the sink can be used to indicate the network conditions and dynamics. As shown in Table 4.4, the cPDR of CTP+Drip and CTP+OSR were 95.96% and 96.22%, respectively. Both are better than the baseline. The node distribution in terms of average hop counts is shown in Figure 4.5. In CTP+Drip experiment 48% of the nodes had an average path length more than 4 hops away from the sink, whereas for CTP+OSR it was 27%. The results indicated that the network

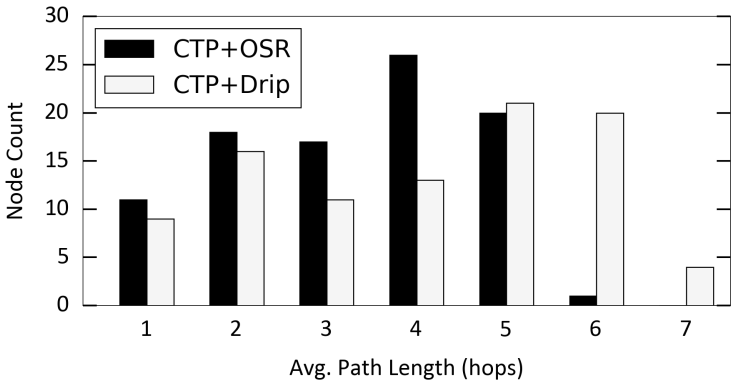


Figure 4.5. Nodes distribution for low power WSN test of CTP+Drip and CTP+OSR in Indriya.

condition during the experiment time period of CTP+OSR was better than that of CTP+Drip.

Table 4.4 lists the comparison between OSR and Drip. Drip achieved 100% downstream packet delivery ratio dPDR due to its flooding nature, but with the TxCost of 97.40 per packet. OSR, on the other hand, achieved a dPDR of 97.50% with the transmission cost of only 5.54 per packet, more than 17 times less than that of Drip. The result demonstrates that OSR is much more energy efficient. In Drip, the dissemination process is global and each new packet would reset the Trickle timer to the minimal interval. Hence its transmission cost is mainly determined by the new packet rate, node density, and network size [43].

Regarding duty cycle, Drip has largely increased the duty cycle, compared to the baseline, from 3.32% to 5.13% (i.e., a 54.54% increase). Due to the better network condition during the CTP+OSR experiment (e.g., higher cPDR and smaller height of the collection tree), it resulted in even a better duty cycle than the data collection baseline. The results basically indicated that OSR had negligible impact on node's duty cycle.

Table 4.4.
Comparison between OSR and Drip on Low Power Indriya Testbed with 95 Nodes

	Network Condition	dPDR (%)	TxCost	Duty Cycle (%)
CTP	cPDR: 93.47% Max. distance: 7 hops	–	–	3.32±1.30
Drip	cPDR: 95.96% Max. distance: 7 hops	100	97.40	5.13±1.19
OSR	cPDR: 96.22% Max. distance: 6 hops	97.50	5.54	2.78±1.26
OSR_{oh}	cPDR: 95.23% Max. distance: 6 hops	99.24	14.37	2.78±1.26

The Effect of Opportunistic Routing

We found that for most of the nodes in the Indriya testbed, the size of the parent set was 1 using CTP as the underlying collection protocol due to the relatively indoor stable environment. To further investigate the effect of opportunistic routing in the Indriya testbed, we examined the OSR overhearing option (denoted as OSR_{oh}) by extending each intermediate node's direct child set with a maximum of 5[3] snooped children (and $\omega = 1$) as discussed in Section 3.2.5. The experiment used the same application configuration as described above. The result is given in Table 4.4. Despite worse network conditions compared to the OSR experiment, OSR_{oh} has significantly improved the downward PDR (99.24%) with a higher TxCost compared to the basic configuration. However, the TxCost of OSR_{oh} was still about 7 times less than that of Drip and had negligible effects to the total energy consumption, as indicated by the average duty cycle (3.29%).

[3]The maximum number 5 (snooped children) was selected arbitrarily in this experiment. In practice, this number can be adapted to the network condition for better transmission cost without affecting the delivery performance.

Table 4.5.
Statistics of Multicasts for OSR and OSR_{oh} in Indriya

Test	**Total Fwds**	**Fwds by Multicasts**	**Multicasts Caused by FP**	**FP Ratio**
OSR	982	160 (16.29%)	159	99.38%
OSR_{oh}	1496	945 (63.17%)	338	35.77%

By overhearing, it would be possible for a node to add a grandchild into its direct child set, especially in dense network like Indriya. So the grandchild(ren) may cause multicasts in OSR in addition to the false positives. Table 4.5 shows the number of multicasts for OSR and OSR_{oh}. For the basic configuration of OSR, 99.38% of the multicasts were caused by false positive matched child(ren). However, this did not help for the passive opportunistic routing since each node was likely to have only one parent in its parent set. OSR_{oh} had 4.91 times more multicasts than that of the basic configuration of OSR, among which 35.77% of them were caused by false positives and the rest were caused by overheard grandchild(ren). This also made multicast become the main transmission scheme in OSR_{oh} as 63.17% of the total packet forwardings were transmitted through multicasts in OSR_{oh} test compared to 16.29% in the OSR test.

4.5 Evaluation in TOSSIM

We further conducted more simulations using TOSSIM for much larger network sizes and higher dynamics. We generated two networks of sizes 225 and 400 nodes uniformly distributed in a square area with the sink at a corner. The 400-node network was expanded from the 225-node network on both dimensions, retaining the same node density. The collection cycle interval was 10 minutes. The sink sent a command packet to a randomly picked node every minute starting from the third

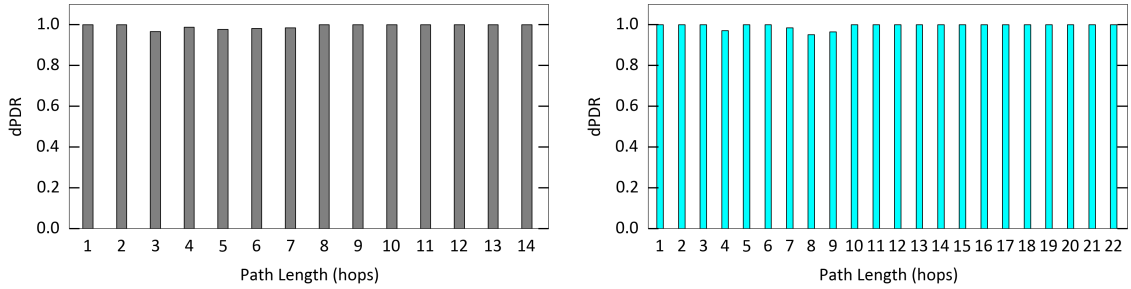
cycle in simulation. The simulation terminated when 600 downstream packets were sent. The *MAX_BFLT_LEN* was configured to 16 bytes and 20 bytes for 225-node and 400-node simulations, respectively. Since TOSSIM is targeted for MicaZ platform, the link layer MTU was configured to 72 bytes to fit in MicaZ RAM space.

Most nodes in the simulations switched their parent nodes rapidly. On average, most nodes have more than 3 parents in their parent set with in a time window of 4 collection cycles, hence it is not necessary to use overhearing to extend the parent set. The OSR implemented the optimization with adaptive maximum of multicasts. The maximum retransmission for each multicast was adaptive to the average link packet reception rate between a node and its children, as described in Section 3.2.5.

Table 4.6 shows the overall performance of two experiments of 225 nodes and 400 nodes, respectively. The data collection cPDR was 99.93% and 99.18% for 225-node and 400-node network, respectively. The maximum node's average path length to the sink was 9 hops and 16 hops, respectively. Note that due to network dynamics a node's actual path length can be much longer than its average path length. Both tests achieved dPDR above 98%, which has not been affected by the expansion of the network size. The per packet transmission cost has significantly increased compared to the basic OSR test in Indriya due to much more dynamic network conditions, longer paths, and much more multicasts, as indicated by Table 4.7. The link unicast retransmission ratio for both tests were around 50%, indicating a noisy and dynamic network condition. Overall, adaptive multicast has saved 12.69% and 14.08% total transmissions for 225-node test and 400-node test, respectively. Regarding opportunistic routing, 3.33% and 4.33% of the packets experienced at least one active opportunistic routing occurrences (due to unicast delivery failure) in the 225-node simulation and 400-node simulation, respectively. On the other hand, both tests resulted in much more passive opportunistic routing occurrences (due to multiple matched children) than the active ones. 19.66% packet forwardings were transmitted through multicasts in 225-node experiment, whereas in 400-node it was 33.30%. For all the multicasts, 60.99% and 51.31% in 225-node test and 400-node test, respectively, were caused by

Table 4.6.
OSR Overall Performance in TOSSIM Simulation

	Network Condition	dPDR (%)	TxCost
225	cPDR: 99.93% Max. distance: 9 hops	98.67	11.12
400	cPDR: 99.18% Max. distance: 16 hops	98.96	22.04



(a) 225 nodes.

(b) 400 nodes.

Figure 4.6. Downsward packet delivery rate distributed on downsward path length.

false positive matched child(ren). The effect is that for 225-node network, 13.33% of the downward packets have experienced at least one instance of passive opportunistic routing, whereas for the 400-node network it was 36.33%, about 3 times of that compared to the 225-node test, due likely to the larger network size and the longer downward path length. The occurrences of the opportunistic routing introduced about 15% $\sim$ 18% duplicate traffic compared to the traditional source routing, which is inevitable due to the probabilistic nature of the Bloom filter (e.g., 9% $\sim$ 50% duplicate traffic as reported in ORPL Figure 6(d) [25]).

Figure 4.6 shows the dPDR based on the downstream path length. As we can see, the packet delivery performance of OSR scales well as the downstream path length increases.

Table 4.7.
Detailed Statistics of OSR in TOSSIM Simulation

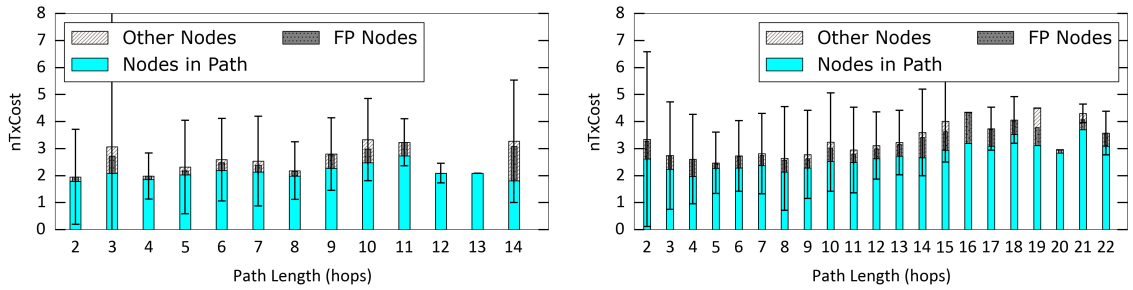
Size	**225**	**400**
Total Fwds	2986	5039
Fwds by Multicasts (% Total Fwds)	587 (19.66%)	1678 (33.30%)
Multicasts Caused by FP (% Total Multicasts)	358 (60.99%)	861 (51.31%)
Ucast Retx	49.15%	51.69%
Pkt. with Active Opp.	3.33%	4.33%
Pkt. with Passive Opp.	13.33%	36.33%
Duplicate Traffic[a]	15.32%	17.92%

[a]With respect to link layer transmissions.

4.5.1 Normalized Transmission Cost

In the following we normalize the performance metric using the path length (actually the path length - 1, as the sink's transmissions are not considered) to provide path length independent metric for evaluation, which actually measures the scalability as a function of the hop count. For instance, the ideal TxCost to deliver a 4-hop source-route packet and an 8-hop source-route packet is 3 and 7, respectively, excluding the transmissions from the sink. After normalization using the path length - 1, the normalized transmission costs (nTxCost) of both packets are 1, which indicates a flat linear scale factor as hop count increases.

Figure 4.7 shows the average nTxCost to deliver downstream packets distributed on the source-route lengths. The average nTxCost of 225-node and 400-node tests were 2.57 and 3.32, respectively. The higher nTxCost of 400-node test was mainly due to the worse link conditions (e.g., more interference and congestion) and higher number of multicasts during in the experiment. The nTxCost increased as the downstream



(a) 225 nodes.

(b) 400 nodes.

Figure 4.7. Normalized transmission cost distributed on downward path length.

packets were sent to the nodes in the central area of the network (i.e., downstream path length of 8∼10 hops in 225-node network and 12∼16 hops in 400-node network), which had larger size of direct child sets. The nTxCost then maintained as the path length became longer. This is because the longest paths (i.e., path length longer than 11 hops in 225-node network and longer than 16 hops in 400-node network) traveled to the edge area of the network, where nodes had very few children in their direct child set. As we will show later, the nTxCost is strongly related to the direct child set size and hence the number of multicasts. Overall, the nTxCost is scalable as the downstream path increases.

For each downstream packet, its transmissions come from two sources: the source-route nodes and other nodes (i.e., duplicate traffic). Other nodes may participate in the dissemination when it is a false positive receiving multicast or if it receives an active opportunistic routing request and has direct child(ren) in the path Bloom filter. As inlucded in Table 4.7, 15.32% of the transmissions in 225-node test were from the other nodes, among which 11.1 percentage points are from false positive nodes. For 400-node test 17.92% of the transmissions were from other nodes and 14.17 percentage points of them are from false positive nodes.

4.5.2 The Size of the Direct Child Set

In this section we examine the size of the direct child set of nodes distributed on their path length from the sink. Note that due to network dynamics a node's actual path length can be longer than its average path length.

Figure 4.8 shows the average size of the direct children set of node distributed at each hop for both experiments. Nodes near the sink and in the central area of the network had more children compared to the nodes with larger path lengths. On average, a node in 400-node network had 4.14 children with a maximum of 12 children. In 225-node network, on average a node had 2.28 children with a maximum of 9 children. The result demonstrated the scalability of OSR with respect to the local direct child set.

The 400-node network was expanded on the basis of the 225-node network, so the nodes in the border area of 225-node network were then in the inside area of the 400-node network. Therefore, these nodes in 400-node network would certainly have larger child sets compared to the nodes in the 225-node network, although they were located at the same area relative to the sink.

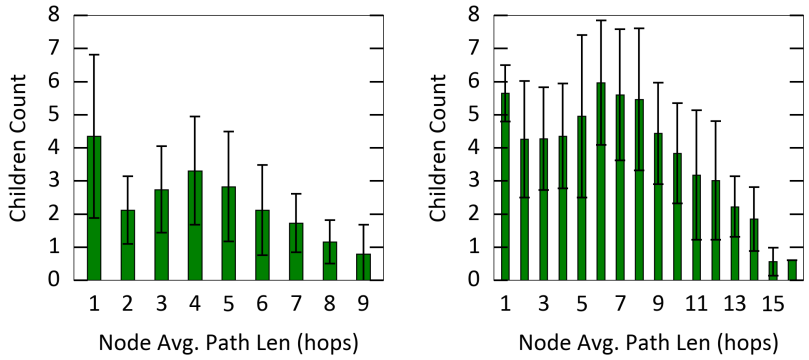
A larger child set size indicates a higher chance to have multiple matched children. Figure 4.9 shows the number of multicasts averaged on nodes' forwarded downward packet, distributed on their average path length to the sink, as computed according to the following equation:

$$Multicast_h = \frac{1}{|N_h|} \sum_{c \in N_h} \frac{Multicasts_k^h}{fwd_k}, \tag{4.1}$$

where N_h is the set of nodes with average path length of h hops, k is a node belongs to N_h, fwd_k is the number downstream packets forwarded by node k, $Multicasts_k^h$ is the number of multicasts of node k.

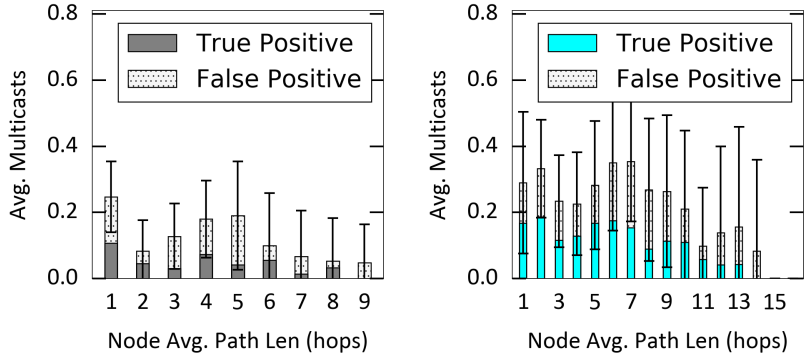
This metric shows how many multicasts a node would have for each forwarded packet, which is related to the size of the node's direct child set. Due to a larger direct child set size, nodes in 400-node network experiment experience a bit more multicasts

than the nodes in 225-node network experiment for each forwarded packet. A larger direct child set also leads to more false positives in multicasts, as a straightforward result of the probabilistic nature of the Bloom filter.



(a) 225 nodes. (b) 400 nodes.

Figure 4.8. Average direct child set size distributed on node's average path length.



(a) 225 nodes. (b) 400 nodes.

Figure 4.9. Average number of multicast occurrences distributed on node's average path length.

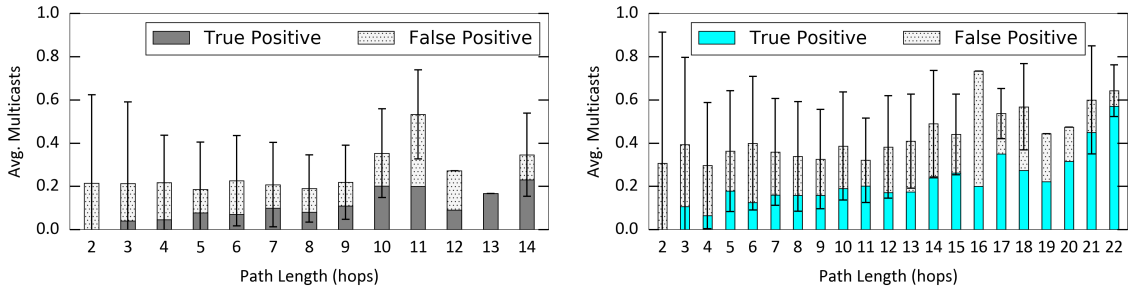
4.5.3 Distribution of Opportunistic Routing

In this section we examine the distribution of the occurrences of the opportunistic routing. Figure 4.10 shows the normalized average number of multicasts for each downstream packet (instead of the node) for 225-node and 400-node network experiments distributed on the downstream path length. The packet in the 400-node network test traveled through the nodes with averagely larger direct child set hence experienced more multicast occurrences. On average, during the downward routing process each packet in the 400-node network experienced 0.37 multicasts at each hop compared to 0.21 multicasts in the 225-node network. The overall multicasts caused by false positive child(ren) is shown in Table 4.7. The number of multicasts increased quickly for packets of path length 9∼11 hops in 225-node network and 12∼16 hops in 400-node network, then deceased as the path length is longer, similar to that of the nTxCost.

Higher number of multicasts indicates more chances of passive opportunistic routing. Figure 4.11 shows the normalized number of opportunistic routing occurs for downstream packet distributed on the path length, which exhibits similar pattern as the number of multicasts. On average, in the 400-node network experiment, each packet experienced 0.09 opportunistic routing at each hop along the downstream source-route, compared to 0.04 for the 225-node network test.

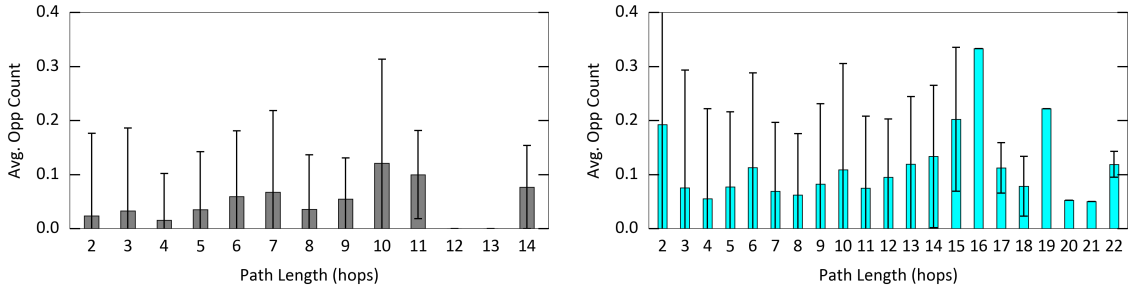
4.5.4 The Effect of Maximum Bloom Filter Size

We conducted more simulations to investigate the effect of the maximum Bloom filter size. We reconfigure the *MAX_BFLT_LEN* to be 10 bytes for both the 225-node and 400-node networks. The result is shown in Table 4.8. As we can see, OSR also achieves the similar high dPDR with a much smaller maximum Bloom filter length. However, due to a higher false positive rate, for the 400-node network with 10 bytes Bloom filter, the duplicate traffic has increased to 37.31%, which then caused more



(a) 225 nodes. (b) 400 nodes.

Figure 4.10. The normalized number of multicast occurrences distributed on downward path length.



(a) 225 nodes. (b) 400 nodes.

Figure 4.11. The number of opportunistic routing occurrences distributed on downward path length.

collisions, as indicated by the higher link unicast retransmission rate. Both the active and passive opportunistic routing occurrences have increased as well.

4.6 A Case Study in ASWP WSN Testbed

In this section, we present a case study of OSR deployment in an real-world out-door heterogeneous WSN testbed in a forest located at the Beechwood Farms

Table 4.8.
OSR Performance with Different Setups of *MAX_BFLT_LEN* in TOSSIM Simulation

Size	**Max. Bflt. (bytes)**	**dPDR (%)**	**nTxCost**	**Ucast Retx**	**Active Opp.**	**Passive Opp.**	**Dup. Traffic**
225	16	98.67	2.57	49.15%	3.33%	13.33%	15.32%
	10	99.33	2.94	51.16%	1.83%	21.33%	17.55%
400	20	98.67	3.32	51.69%	4.33%	36.33%	17.92%
	10	98.50	5.38	58.46%	4.83%	45.17%	37.31%

Nature Reserve (BFNR) of the Audubon Society of Western Pennsylvania (ASWP)[4], Pittsburgh, Pennsylvania, USA [1].

4.6.1 Deployment and Application

The ASWP testbed includes heterogeneous WSN devices that periodically sample the environmental data and transmit to the sink through multihop networking. Starting at August 2017, 94 WSN nodes have been deployed at the testbed, including 42 TelosB, 21 MicaZ, and 31 IRIS motes, with one IRIS sink node. The locations of the sensor nodes are illustrated in Figure 4.12.

As Figure 4.12 shows, the node locations are restricted by the geography and the aesthetic requirements of the nature reserve. The base station can only be placed at the BFNR Nature Center where the Internet access is available, while the sensors are installed around 300 meters away or further. Especially, the number of node locations near the sink are highly limited.

[4]ASWP Beechwood Farms Nature Reserve:
http://www.aswp.org/locations/beechwood/

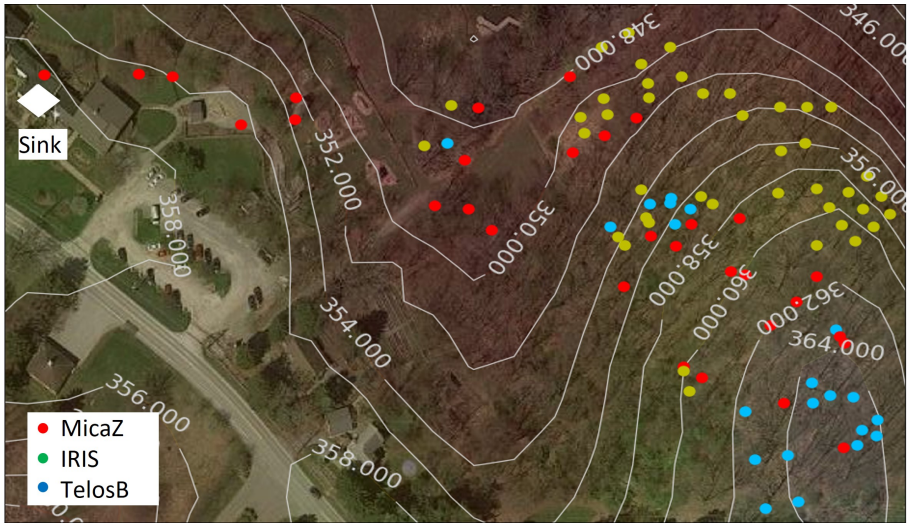


Figure 4.12. Node locations at the ASWP WSN testbed as of August 2017. MicaZ nodes are marked as blue, IRIS nodes are marked as red, and TelosB nodes are marked as yellow.

Thus, nodes are classified into two categories based on their functionalities: *relay nodes* and *regular nodes*. Relay nodes are mainly used for building the backbone from the main locations of interest to the sink node located at the Nature Center; they are only equipped with on-board sensors for voltage, temperature, and humidity. On the other hand, the major environmental data are sampled at the regular nodes which are configured with different combinations of external sensors (e.g., soil moisture, water potential, sap flow, soil temperature) in addition to the on-board sensors. In the ASWP testbed, TelosB and MicaZ motes are mainly used as regular nodes and IRIS motes are preferred as relay nodes due to more powerful transceiver. In total, there are 25 relay nodes and 69 regular nodes. The locations of the IRIS motes are carefully decided, where the nodes are hanging high on the tree and less obstacles sit in between, in order to build a reliable backbone of the network.

The mote application is developed based on TinyOS 2.1.2, which incorporates different components to perform varies tasks including sampling, upward routing [49], downward routing (i.e., OSR), compressed sensing [48, 68], reprogramming [69], and flash access. The data collection relies on the CTP+EER protocol [49] where each

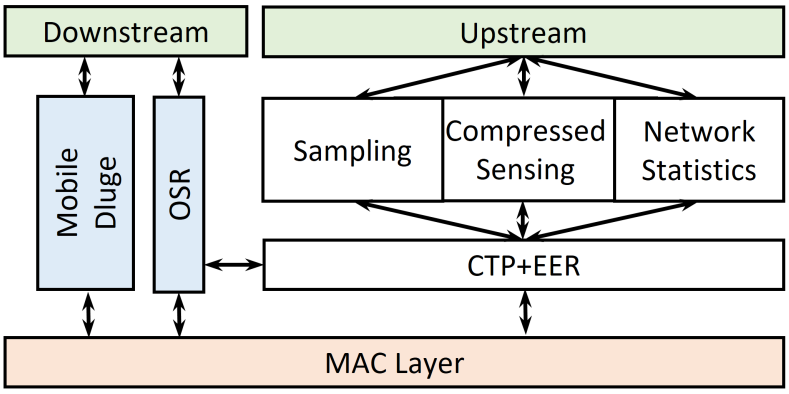


Figure 4.13. Major components of the ASWP application.

node actively explores multiple paths to reach the sink in order for load balancing and global energy efficiency. The motes operate on low lower mode with a sleep interval of 1 second with the sink being always on. The average inter-packet interval is 30 minutes. Figure 4.13 illustrates the major communication components of the ASWP application.

When a packet arrives at the base station, a gateway [70] parses the packet and stores it into a local database, then sends the parsed packet to the INDAMS management system [71] which can be used to monitor the network status in real time.

4.6.2 Protocol Evaluation

OSR was implemented to work with CTP+EER which shares the same public interfaces as CTP. The *MAX_BFLT_LEN* was 16 bytes and the maximum direct child set was 20. In ASWP testbed, the nodes are densely deployed at the main area of interest. Even with the 20-node direct child set, we found some nodes have their direct child set filled up since CTP+EER actively explores the parent set of each node. Thus, we further modified OSR's forwarding scheme: when a node's direct child set is full, it would broadcast even if none of its stored children is matched in the path bloom filter.

The sink sends downward packets on demand with the minimal downward packet interval being 1 minute, in order to minimize the interference to the normal data collection. Upon the reception or forwarding of an OSR packet, the node would start to send its OSR related statistics to the sink following the data collection path. During the experiment period, the network data collection PDR is about 60%∼75% due to heavy vegetation growth. In total 79 downward packets were sent.

Table 4.9.
Performance of OSR in ASWP Testbed

PDR	84.81%
TxCost	12.38
Total Fwds	401
Fwds by Multicasts (% Total Fwds)	116 (28.93%)
Multicasts Caused by FP (% Total Multicasts)	79 (68.10%)
Ucast Retx	28.53%
Active Opp. Count	6
Passive Opp. Count (% Total Fwds)	38 (9.47%)
Duplicate Traffic	53.58%

Table 4.9 shows the overall performance of OSR in the testbed. The unicast retransmission rate is about 28%, indicating a lossy network environment. OSR achieved a PDR of about 85%. Due to the dense deployment and large direct child set, about 29% packet forwardings were transmitted using multicast, among which 68% were caused by false positive children. Passive opportunistic routing occurs about 10% of the total packet forwardings. As a result, the duplicate traffic is about 54%. Overall, the performance of OSR is slightly worse than that in the simulation

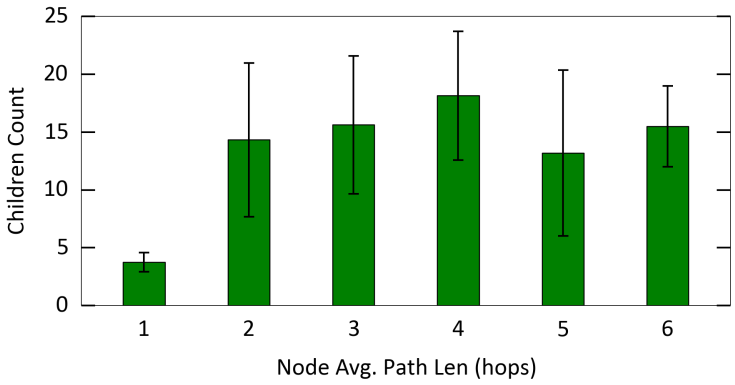


Figure 4.14. Average direct child set size distributed on node's average distance in ASWP testbed.

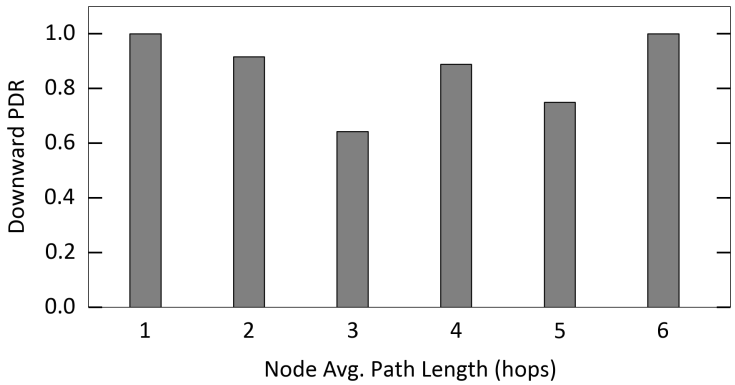


Figure 4.15. Downward packet delivery ratio distributed on node's average distance in ASWP testbed.

and the indoor testbed. One of the main reason is the sever link asymmetry caused by hardware heterogeneity, as described in Chapter 6. The other reason is the very dynamic out-door environment causing the link conditions very unstable.

The size of the direct child set is illustrated in Figure 4.14. Compared to Figure 4.8, the nodes in the ASWP testbed have a much larger direct child set due to its dense deployment and the adoption of CTP+EER. On average, each node has 15 children in its direct child set. This is the main reason of the large percentage of multicast transmissions in ASWP testbed.

Figure 4.15 shows the downward PDR based on the node's average distance from the sink. The PDR is worse when the downward packet is sent to the main body of the network, which may likely due to more collisions in that area.

4.7 Discussion

OSR has achieved reliable delivery of the downward unicast packets and desirable scalability as the network diameter and network size increase. OSR (with CTP) has also shown to achieve better energy efficiency compared to TinyRPL and Drip (with CTP) implemented on the same TinyOS platform. In this section we discuss our insights, and the limitations of current OSR implementation.

Scalability. Through our comprehensive evaluations, OSR achieves significantly better scalability compared to RPL storing and non-storing modes on two implementations and ORPL. OSR enables a very small and localized routing table compared to RPL storing mode and ORPL; simultaneously, OSR compresses the source-route effectively with respect to RPL non-storing mode. Therefore, OSR provides desirable scalability for resource-constrained real-world WSN/IoT deployments.

Opportunistic Routing. OSR depends on the upward traffic to build the child/parent set. Thus, if the collection protocol is the best-path oriented, OSR may not be able to offer significant opportunistic routing due to the lack of potential helper forwarders in a relatively static communication environment. Even though in such a situation, OSR would degrade back to the traditional source routing in routing perspective, its compression of source-route in packet header is still exactly effective. OSR could benefit by working with a load balanced data collection protocol (e.g., CTP+EER [49]), which actively switches parents to balance the traffic load hence expands the parent set, in the case of static network condition. On the other hand, opportunistic routing in OSR introduces duplicate traffic to some degree due to its probabilistic nature.

Link Asymmetry. To build any downward path, OSR relies on upward routing tomography, whereas RPL non-storing mode uses the DAO messages to maintain the network topology at the root. Both approaches may lead to suboptimal downward path selection due to the link asymmetry.

Interaction with IP. We implemented OSR protocol, working with CTP in TinyOS, to validate our OSR approach for downward routing scalability and reliability. Thus, our current implementation lacks the ability to interact with IP like that of RPL and its variations. However, the principles of OSR can be readily applied to RPL non-storing mode to extend and improve its capability, which is considered in our future work.

Differences from ORPL/CBFR. OSR uses the Bloom filter to encode the source-route, whereas both ORPL and CBFR uses Bloom filter to compress the subgraph at each intermediate node. Each intermediate node stores its direct child set in OSR versus the subgraph of descendants in ORPL/CBFR. Also, OSR constructs the direct child set easily by intercepting the data collection packets, which is strictly localized and has negligible overhead. CBFR also intercepts the data collection packets, however, like ORPL, it requires the nodes in the network to exchange their Bloom filters to gather the subgraph information, introducing additional transmission overhead. Moreover, OSR can, to some extent, turn false positive into an opportunity in its passive opportunistic routing via local multicast. In CBFR, transmissions are broadcast based to explore all the possible downward paths to reach the destination node, resulting in a high transmission cost. On the other extreme, ORPL opportunistically selects only one node at a time for downward routing in the situation of multiple matches, hence it risks of delivering the packet to a false positive target node. An additional false positive recovery scheme is required in ORPL to address this problem, with increased delay. In contrast, OSR either opportunistically multicasts to all matched nodes, or opportunistically broadcasts when the downward link is broken. Hence, OSR is much less aggressive than CBFR, and has no need for false positive

recovery in comparison with ORPL. The packet and duplicate suppression in OSR ensure that the packet forwarding by false positives would not chain forever.

ASWP Case Study. The case study of OSR in the ASWP testbed has shown worse performance (e.g., lower dPDR and higher TxCost) than the experiments in simulations and indoor testbed. Several factors may affect the OSR's performance. First and foremost, the environment of the ASWP deployment is much worse than the indoor testbed due to the growth of vegetations in the forest. Secondly, the topology of the ASWP testbed may limit the effect of the opportunistic routing in the first two hops. As Figure 4.12 shows, the nodes at the first two hops do not have enough parents for opportunistic routing. The nodes at the second hop are actually the bottleneck of the network, for both data collection and dissemination. The downward packet could not be 100% delivered to the second hop nodes, which also affects the delivery for the rest of the network. Thirdly, the problem of link asymmetry is severe in the ASWP testbed due to the heterogeneous sensor node platforms, as will be presented in Chapter 6. This may lead to worse downward path selection compared to the homogeneous networks in the Indriya testbed. Finally, the ASWP testbed is densely deployed at the main area of interest, which may generate more collisions when delivering the downward packets.

5 MOBILE CODE DISSEMINATION FOR WIRELESS SENSOR NETWORKS

Network maintenance becomes a key issue in long-term WSN deployments. Among other tasks, such as replacing batteries and fixing broken nodes, network maintenance also involves reprogramming currently deployed nodes for updating and improving applications. Manually reprogramming sensor nodes is very cumbersome as it requires retrieving the nodes from their deployed locations. To address this problem, many approaches (e.g., [11], [32–34], [72–77]) have been proposed in the past years for supporting over-the-air programming (OAP) through wireless communications. Most of them have been thoroughly examined through simulations and laboratory experiments. However, real-world WSN deployments usually have some unique features which are very challenging to the existing OAP approaches.

First, heterogeneity becomes a common scenario in WSN deployments, where multiple node platforms (e.g., MicaZ, IRIS, TelosB), sensors, and applications may coexist on the same WSN testbed. From this arises the need for point-to-point or subset reprogramming in WSNs. However, whereas very few studies (such as [32], [74]) support point-to-point/subset reprogramming, most existing approaches such as Deluge [11], MOAP [33], MNP [34], CORD [72], Zephyr [73], the work in [75], EasiLIR [76], and ROLP [77], disseminate the code image to all the nodes in the network. Such a dissemination approach simply fails for heterogeneous WSN deployments with multiple node platforms, where different code images are required for different subsets of nodes.

Secondly, real-world outdoor WSN deployments such as [78], [79], and [4] usually work over low-power link layers for better energy efficiency, such as the typical low-power-listening (LPL) mode in TinyOS [36]. Sleep intervals in LPL mode largely extend per-packet delivery time. Since reprogramming usually involves bulk

Table 5.1.
Performance of Deluge with LPL and Always-on

	LPL	**Always-on**
ADV Pkts	25626	18
REQ Pkts	64	3
DATA Pkts	11792	141
Total Pkts	37487	162
Completion time (seconds)	297.97	4.9
Estimated energy consumption (mAs)	11564.21	111.72

code image dissemination, the total delay significantly degrades the performance of the previous approaches [11], [32–34], [72–77]. While the recent work of ROLP [77] addresses this problem by dynamically adjusting the sleep intervals during image dissemination, ROLP still disseminates the code image to the entire network, which fails to reprogram any heterogeneous WSN.

We present an illustration example of how low power affects the performances of the approaches designed for always-on links. We use standard Deluge to disseminate a simple Blink program of 2592 bytes (3 pages or 141 packets) and compare the performance under LPL mode and always-on mode. The result is summarized in Table 5.1. With LPL mode, Deluge transmits about 200 times more packets than those over always-on links. It is about 50 times slower and consumes about 100 times more energy. This result clearly shows that LPL has dramatically degraded the efficiency of Deluge, as also indicated in [77].

Long-term outdoor WSN deployments would usually require periodic on-site maintenance visits (e.g., battery replacement and faulty node fixing) to keep the network operating in a healthy and sustainable manner [78]. In view of this, we take a new

approach to simultaneously address both challenges described above. We introduce a novel concept of mobile code dissemination, and present MobileDeluge, a general mobile network-reprogramming tool based on Deluge. Equipped with a gateway laptop and a base station, as shown in Figure 5.1, MobileDeluge is a hand-held code dissemination tool for outdoor WSN deployments over low-power links. It enables wireless reprogramming of WSN nodes in harsh but accessible environments within a one-hop neighborhood with respect to the hand-held Deluge base station. MobileDeluge creates a control service to coordinate the mobile Deluge base station and the target sensor nodes within the neighborhood of the mobile Deluge base station for code dissemination. The key idea is to establish an instant connection between the mobile Deluge base station and its target sensor nodes within the neighborhood, where the target nodes are to be updated with the same new code image. Once the connection is established, the target nodes are asked to switch to a different channel and disable LPL so that they can be reprogrammed efficiently. Since MobileDeluge can be brought close to the target nodes when reprogramming is needed, the reprogramming is limited to a single-hop neighborhood. In this way, MobileDeluge enables a significant amount of energy savings at intermediate nodes compared to traditional multihop code dissemination approaches.

5.1 Related Work

Earlier network programming protocols usually distribute the entire new code image to the network [11], [32–34], [72]. Crossbow Network Programming (XNP) [32] was known to be the first network reprogramming protocol designed for WSNs. It operates with TinyOS, disseminating the whole code image to the nodes within a single hop network. multihop Over-the-Air Programming (MOAP) [33] supports multihop network programming by employing a neighborhood-by-neighborhood transport mechanism called Ripple to distribute the new code to the whole network, and a simple sliding window method to track and manage retransmissions. multihop Network

Figure 5.1. MobileDeluge, a hand-held mobile code dissemination tool.

Programming (MNP) [34] presents an efficient code dissemination mechanism by reducing the message collision and hidden terminal problem, attempting to guarantee that the node with the most impact in a neighborhood is selected to be the only source that transmits the new program. Deluge [11] is the *de facto* standard code dissemination protocol in TinyOS. It uses Drip protocol to control the code dissemination process. In Deluge, the code image is divided into a set of fixed-size pages, enabling spatial multiplexing so that large data objects are efficiently disseminated over a multihop network. CORD [72] delivers the code image to a subset of nodes called core nodes, and then the core nodes act as the source and disseminate the code image to their neighbors.

More recently, research efforts focus on incremental reprogramming approaches to reduce the transmitted data size, hence saving energy. Such approaches follow a common pattern of computing the difference between old and new code images, transferring this difference, and locally rebuilding the new program at the nodes.

Zephyr [73] performs a byte-level comparison between the old and new program binaries using an optimized version of the Rsync algorithm [80]. To reduce the size of the delta between the old and new programs, it applies application-level code modifi-

cation, mainly in the function call indirection, for compensating the effects of function shifts caused by program modification. Different from Zephyr, the work of [74] uses block-level Rsync comparison algorithm to compute the differences between the old and new code images. As for dissemination, it uses BLIP IPv6 stack as the under layer routing protocol in supporting point-to-point multihop code distribution. At the node side, it uses a Deluge-like volume management to rebuild the new program. In [75], the longest common subsequence between old and new code images is computed using Hirschberg's algorithm [81] at a byte level. It builds an edit map specifying the edit sequence required by a node to transform the running program into a new program. It further applies a heuristic-based optimization strategy to encode the edit map to reduce the transmitted data size. EasiLIR [76] presents an energy efficient incremental wireless reprogramming scheme, which avoids read/write operations on nonvolatile memory (e.g., external flash memory) as much as possible. It applies in situ modification which directly modifies the binary code stored in memory to create the new program without entirely rebuilding the program. In case of large modifications that may break the program time constraint, it also applied a lightweight segmented rebuilding for directly creating a new image in memory.

To efficiently reprogram WSNs operating over low power links, ROLP [77] modifies the three-way handshaking scheme in Deluge. The idea is to synchronize the low power settings in a neighborhood through the exchanging of augmented Deluge control packets. When there are data to be sent, both sender and receiver nodes wake up to always-on states (i.e., set sleep intervals to 0). On the other hand, when the transmission is finished, the nodes go back to LPL mode again, tuning the sleep settings according to the neighbor nodes' information.

5.2 Design of MobileDeluge

In this section, we present the design of MobileDeluge. MobileDeluge is a general network reprogramming tool built based on Deluge and effectively addresses both

reprogramming challenges for long-term WSN deployments: heterogeneity and operating on LPL mode.

5.2.1 MobileDeluge Outline

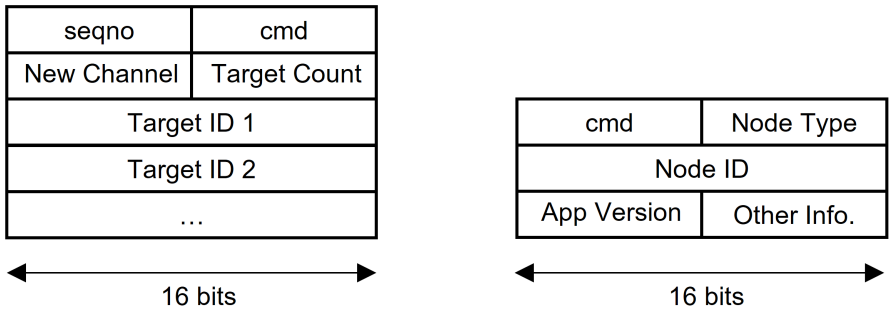
Our design of MobileDeluge has the following key features: 1) one-hop network reprogramming, so that the reprogramming of a multihop network will be achieved by its mobility; 2) a novel control service enabling the retrieval of the platform information of the nodes in a one-hop neighborhood of the MobileDeluge base station (referred to as MobileBase), so that only the target nodes of the same platform type are reprogrammed at a time to address the heterogeneity; and 3) both MobileBase and the target nodes switched to a different channel with LPL disabled, allowing the fast and efficient transmission of the new code image without the interference to the rest of the network. In the following subsections, we present our design in detail.

5.2.2 Subset Reprogramming

MobileDeluge uses the basic broadcast scheme to establish the connection between the MobileBase and the target nodes, which limits its working range to a single hop. Its logic is split into two parts: the MobileBase side and the node side.

MobileBase

The MobileBase acts as a bridge between the target nodes and the mobile computer gateway connected to it. It receives commands from the gateway, and then broadcasts to the nodes. We divided the commands into two sets. The first set is the regular Deluge commands, which is directly processed by the standard Deluge logic. The other set, referred to as Mobile commands, is used for communication between the MobileBase and the target nodes. Basically two Mobile commands are defined: DISS and ABORT. DISS command starts a reprogramming cycle by notify-



(a) Command packet structure. (b) Reply packet structure.

Figure 5.2. MobileDeluge packet structure.

ing the target nodes to get ready, whereas ABORT is used to stop the reprogramming and reset the target nodes to their original state. In the Mobile command packet, the command occupies one byte, the other fields in the packet are the target nodes list, new channel, and other auxiliary information. The command packet structure is shown in Figure 5.2(a).

Before a reprogramming cycle is started, the target nodes are operating on the original channel with LPL enabled. In order to notify the target nodes, the MobileBase starts issuing a DISS command, which is broadcasted in the original channel of target nodes with LPL enabled. It then waits for the replies on the original channel. If all the nodes replied without delay, the MobileBase switches to a new channel and disables LPL. Otherwise rebroadcasting is needed until the maximal number of retransmissions is reached. When the MobileBase and the replied target nodes are on the new channel and over always-on links, regular Deluge commands will be issued to complete the reprogramming.

If the target nodes are mistakenly selected, or for some reason the reprogramming is no longer needed, an ABORT command can be issued to reset the nodes. The ABORT command does not require nodes' reply so that the nodes can reset as soon as the command is received. Instead, it is broadcasted for multiple times to ensure reliable delivery. Note that, when an ABORT command is needed, the target nodes

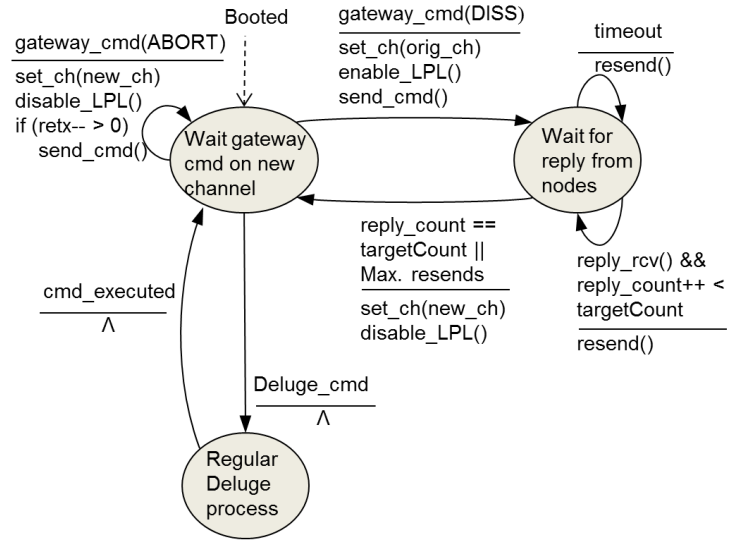
are in the new channel. Thus, the MobileBase has to stay or switch to the new channel for broadcasting, depending on its current state. The MobileBase side's control is presented in Figure 5.3(a) as a finite state machine.

Node Side

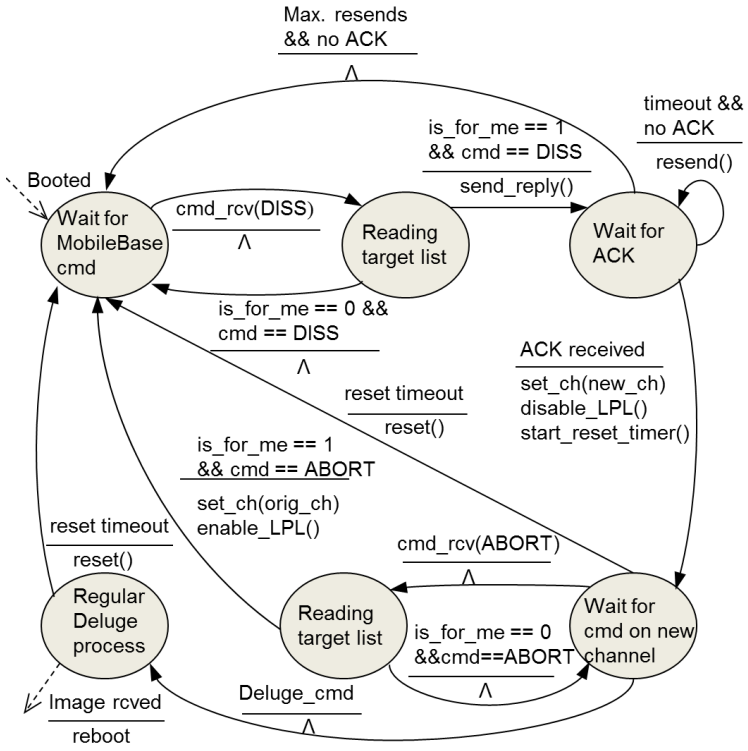
Due to the broadcasting nature of wireless communications, a node's transmission can be received by any other nodes in its neighborhood. When a node operating in the regular application (i.e., the original state) receives a Mobile command packet, it checks the target list in the command packet. If it is not in the target list, the command is ignored. Otherwise, it responds according to the types of the command. If a DISS command is received, it sends a reply to the MobileBase and waits for an acknowledgment. If the reply packet is acknowledged, it switches to the new channel, and disables LPL, getting ready for reprogramming; otherwise, if no acknowledgment is recieved after several retransmissions, it ignores the command. On the other hand, if an ABORT command is received, it resets to the original state (i.e., switches to the original channel and enables LPL) immediately.

When a node switches to the new channel, it waits for Deluge commands to finish the reprogramming. A reset timer is started to reset the node to the original state if the reprogramming is not finished in a certain time. The node side's control is presented in Figure 5.3(b).

Figure 5.4 shows an example of packet exchanges at the start phase of a reprogramming cycle. MobileBase broadcasts DISS command with target list (A, B) and new channel 15 in the original channel with LPL enabled. All nodes, A, B, and C, will receive the command. But only node A and B will send a reply after checking the target list. If the reply is lost, a retransmission is triggered (e.g., at node B). When the acknowledgment is received, the nodes switch to the new channel and disable LPL. On the MobileBase side, when all replies are received, it switches to the new channel and disables LPL.



(a) MobileBase control logic.



(b) Node side control logic.

Figure 5.3. Finite state machines of MobileDeluge control logic.

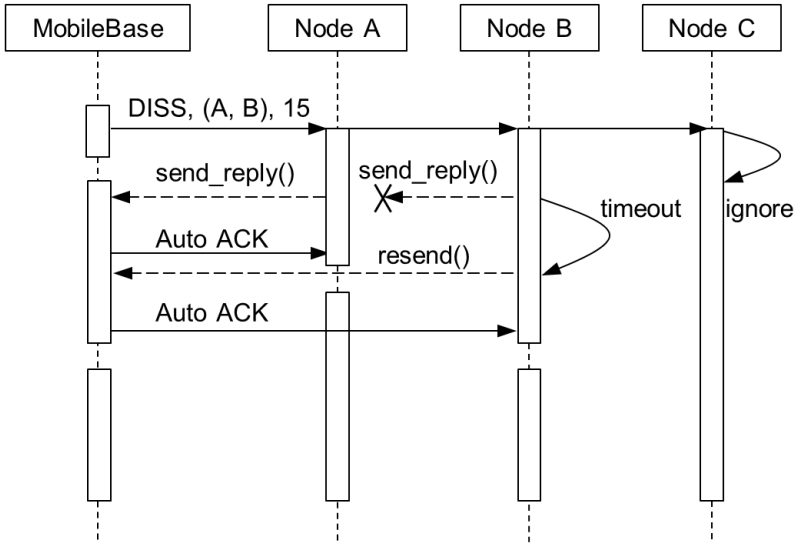
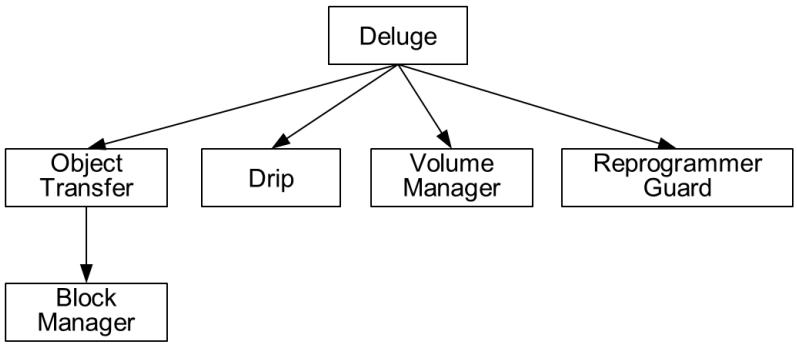


Figure 5.4. Start phase message exchange.

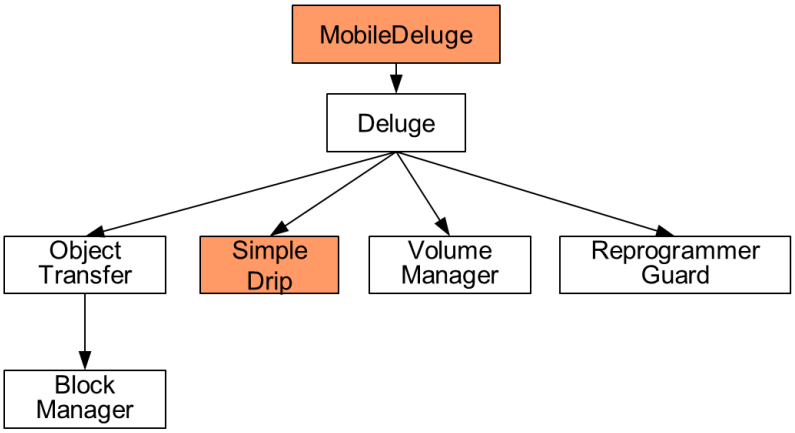
SimpleDrip

Since the reprogramming is limited to a single hop in our design, we replaced the multihop dissemination protocol Drip with a simplified version, referred to as SimpleDrip. SimpleDrip is a one-to-many single-hop dissemination protocol that maintains the same interfaces as Drip so that any protocols depending on Drip can seamlessly use it if single-hop dissemination is preferred. In the MobileDeluge, SimpleDrip replaces Drip to disseminate the Deluge commands to the target nodes in the neighborhood. Different from Drip, where every node periodically broadcasts according to the Trickle timer, in SimpleDrip, sender's (e.g., usually a base station) behavior and receiver's behavior are separate. The sender broadcasts the packet containing the new value that follows a linearly increasing timer, whereas receiver nodes do not transmit any packets. Thus, the rest of the network experiences no Drip traffic as all target nodes are receivers with the only sender being the MobileBase.

In summary, the code structure of Deluge and MobileDeluge is shown in Figure 5.5(a) and Figure 5.5(b), respectively.



(a) Deluge.



(b) MobileDeluge.

Figure 5.5. Code structures of Deluge and MobileDeluge.

Mobile Gateway

We develop the MobileDeluge gateway software, which runs on a laptop and controls the reprogramming cycle. It integrates the Mobile commands and the Deluge commands. MobileDeluge hence can form as a potential generic mobile command/query system for WSNs, with some extensions. Currently only two commands are implemented, as described above. When a node receives the DISS command, it can send very useful information along with the reply message, such as platform type, application version number, or neighbor table, to the gateway through the MobileBase. Target platform information is significant in heterogeneous WSNs, since mistakenly reprogramming a node using the code image for a different platform can

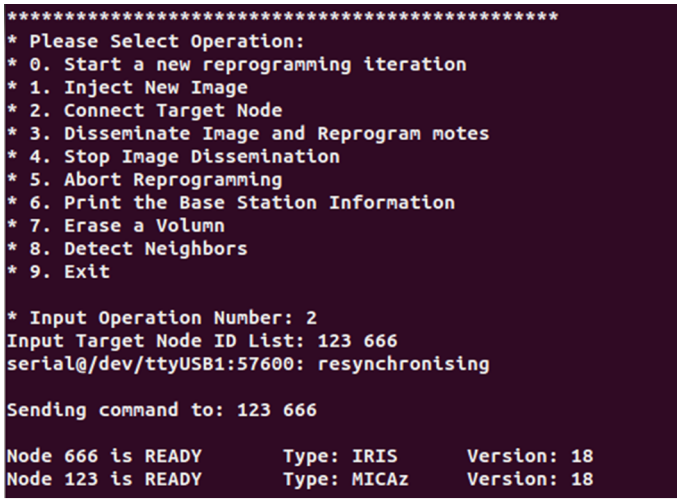


Figure 5.6. An illustration of MobileDeluge gateway interface, showing the platforms and application versions of target nodes.

crash the node. In addition, version information tells whether reprogramming is necessary for the node. Figure 5.2(b) shows the reply packet structure. Figure 5.6 shows an example of the MobileDeluge gateway interface.

5.3 Evaluation

We implemented MobileDeluge based on Deluge in TinyOS. In this section, we examine the performance of MobileDeluge in comparison to Deluge on a single-hop network over low-power links. However, to better understand the results, the performance of Deluge over always-on links is also provided as a reference. We shall compare the completion time and number of transmitted packets of the reprogramming process of both mechanisms. We used a sniffer [82] to record all the packets sent by the nodes. The sniffer attached a timestamp to each received packet, which is used to calculate the completion time. In this section, we present our evaluation not only based on laboratory experiments but also including actual reprogramming experience in our real-world ASWP testbed.

Table 5.2.
Program Image Size

	With Deluge	**With MobileDeluge**
ROM (bytes)	43638	43568
RAM (bytes)	3362	3470
Image size (bytes)	44544	44544
No. Pkts	660	660

5.3.1 Lab Experiments

Our ongoing WSN testbed study includes a combination of a basic data-collection service, CTP instrumentation for network management and analysis, routing inference, and network reprogramming with MobileDeluge. In order to accommodate all the information into one packet, we increased the *TOS_DATA_LENGTH* (i.e., the MAC layer payload size) to 75 bytes. Based on different sensor measurement requirements, there are six versions of programs differing in the configuration of parameters. Table 5.2 presents the image size of the testbed application with five external sensors when MobileDeluge or the standard Deluge is used, respectively. The other versions of programs differ in parameter configurations and have similar memory occupation. MobileDeluge occupies about 100 bytes more RAM than Deluge with all the functional augmentations discussed in the previous section, slightly less ROM, and the same image data size.

Reprogramming a Single Node

Table 5.3 shows the number of transmitted packets and the completion time for reprogramming a single node. Deluge with LPL was 21.8 times slower and sent 142 times more packets than MobileDeluge. On the other hand, despite the start phase, which took several seconds and sent 149 Mobile command packets over an LPL link,

Table 5.3.
Comparison of Deluge and MobileDeluge for Single Node

	Deluge&LPL	**MobileDeluge**	**Deluge**
DV Pkts	104852	188	187
REQ Pkts	302	46	44
DATA Pkts	44292	660	660
Start Pkts	0	149	0
Reply Pkts	0	1	0
Total Pkts	149446	1044	891
Completion time (seconds)	1365.95	59.95	54.96

MobileDeluge has very similar behavior of the Standard Deluge, which is expected since once the start phase is finished, MobileDeluge actually works on the Deluge routine.

Reprogramming a Subset of Nodes

To test MobileDeluge on heterogeneous networks, we setup a single hop network containing 5 nodes, in which 3 of them are MicaZ nodes and others are IRIS motes. The result is summarized in Table 5.4. For Deluge, we only used 3 MicaZ nodes, since it only works in homogeneous networks. MobileDeluge has successfully reprogramed all the target nodes in the hetergeneous network. Compared to Deluge over always-on links, it has transmitted about 200 more packets and taken 7 more seconds (which is in start phase) for completion. However, on low power links, Deluge took 24.7 times more completion time and transmits 138 times more packets.

Table 5.4.
Comparison of Deluge (3 Nodes) and MobileDeluge (3 Out Of 5 Nodes)

	Deluge&LPL	**MobileDeluge**	**Deluge**
DV Pkts	111678	202	175
REQ Pkts	596	51	47
DATA Pkts	39916	663	663
Start Pkts	0	174	0
Reply Pkts	0	3	0
Total Pkts	152190	1093	885
Completion time (seconds)	1362.95	52.96	45.96

5.3.2 Testbed Experience

MobileDeluge has been validated through reprogramming a subset of nodes in the outdoor ASWP testbed, as described in Chapter 4. We wirelessly reprogrammed the nodes in three measurement areas in the testbed, moved to a different reprogramming neighborhood at a time, and recorded the cost of disseminating a 50064 bytes' code image to the target nodes. The statistics are shown in Table 5.5. Due to the unreliable nature of wireless communications, the reprogramming statistics for each trial would vary from one to another. Reprogramming several nodes together needs more packets to be transmitted. However, the time consumption is very similar compared to reprogramming a single node. On the field, the size of the target subsets depends on the radio range of the MobileBase and relative locations of the target nodes. Since the code image is very large compared to regular data packets, the dissemination must be conducted in a very reliable manner. Thus, the effective radio range can be smaller than that in regular communication situations.

Table 5.5.
Statistics of Reprogramming in the Field

Target subset size	**Type**	**Total Packets (DATA, ADV, REQ)**	**Completion Completion Time (s)**
1	IRIS	1196	63.95
1	IRIS	1172	74.94
1	MicaZ	1219	87.93
1	MicaZ	1214	65.95
1	MicaZ	1195	72.94
3	IRIS	1257	68.94
3	IRIS	1283	72.94
3	MicaZ	1890	89.93
3	MicaZ	2115	55.96
4	MicaZ	2208	108.92
Total 21 Nodes	–	14749	762.42
Per Node Avg.	–	702.33	36.30

The manual reprogramming procedure starts from getting the enclosure from the tree. Then several screws that seal up the box and fix the node with the acquisition board have to be taken off, and then, the node needs to be attached to a laptop to be reprogrammed. The previous procedure of getting the node has to be reversed after the reprogramming to put the node back to its deployment location. Usually, it takes a whole day to finish reprogramming the nodes above. Experience shows that MobileDeluge has significantly improved the efficiency of the field reprogramming work.

6 NETWORK DYNAMICS AND BENCHMARKING

Many WSNs measurement studies have been published in the past years for data collection WSN deployments, however, they mainly focus on the link level (e.g., [83,84]) or data level (e.g., [85,86]) characteristics of the deployment instead of the network behavior during operation, whereas understanding the network level behavior is critical for protocol design. Moreover, most of the WSN deployments are short-term and homogeneous, whereas heterogeneous network configurations are envisioned to be more general in the future WSN deployments. Recently the research community has realized the severity of lacking a standardized benchmark suite for WSN evaluation [87,88]. This chapter presents an empirical study of the network level behavior of a long-term out-door heterogeneous WSN deployment and works as an early step for WSN benchmark. We first perform a measurement study which includes link level behaviors, topological characteristics, and temporal characteristics, then organize the dataset with full topological information as a WSN benchmark suite.

6.1 Related Work

In the last decades, the research community has conducted numerous WSN deployments to study varies research problems. Some early efforts (e.g., [12–15]) have provided very valuable experiences for deploying real world sensor networks, however, they focus more on the specific applications instead of the network performance. Many other works have conducted measurement studies on WSN deployments. Detailed link level characteristics are presented in works such as [83,84,89]. Works such as [1,85,86,90,91] have focused on the packet delivery performance of the network. While routing is the critical component of any multihop WSN deployment, very few of the studies focus on the routing dynamics of the deployment. [92] has investigated the

routing dynamics based on CTP [7] using the parent change event and has identified key causes of the parent changes. However, it lacks the depiction of the topological dynamics for the network as a whole.

A few public WSN datasets are available, however, they only provide either sensor readings (e.g., SensorScope [15] and IntelLab [93]) or link signal readings (e.g., SING [86]), lacking the support for topological analysis. [87] is believed to be the first attempt to provide the reference benchmark for the WSNs community. It described the design elements for a testbed infrastructure based benchmark suite, where users only need to provide the protocol firmware and the benchmark would take care of generating the test scenario, such as traffic load, the level of controlled interferences, the test application logic, and performance metrics. [88] presents the recent progress of the benchmark. However, the benchmark is still under development and most likely it would be indoor or near main building infrastructure where infinite power supply is available.

6.2 Data Source

The data is collected in the out-door heterogeneous ASWP WSN testbed described in Chapter 4. In the outdoor testbed, nodes usually die at different rate due to different battery conditions, hardware issues, or workloads. Thus, it is difficult to collect complete data from the full site for our benchmarking purpose. Instead, a subset of 73 nodes are selected consists of 13 MicaZ motes, 24 IRIS motes, and 36 TelosB motes. Each node sends three types of packets regularly serving different purposes:

- **Type I**. Sensor packet. The packet with sensor readings, such as temperature, humidity, soil moisture, and so on, which is transmitted at a random timer interval within 15$\sim$45 minutes, with an average interval being 30 minutes.
- **Type II**. Summary packet. Contains information for network diagnosis, such as the number of parent changes, the number of retransmissions, the number

of dropped packets, and so on. This type of packets are generated on averagely every 2 hours.

- **Type III**. Compressed sensing packet. An experimental packet contains compressed sensor readings of temperature, humidity, soil moisture, and so on, as described in [68]. A timer and a probability threshold controls the generation of the packet. The timer fires in the same way as the Type I packets, however, the packet is transmitted based on a probability ranging in 10%∼20%. For instance, if the probability is 20%, this type of packets are transmitted 20% of the time compared to the Type I packets.

For all the packets, the packet header includes the link information between the source node to its first hop parent, as well as the path information from the source node to the sink. Specifically the following fields are included:

- Link RSSIs. We consider bidirectional links. Hence, each observed link is associated with two RSSI values. (1) uplink RSSI (U_RSSI) is measured at the parent node when a data packet is received; (2) downlink RSSI (D_RSSI) is measured at the child node through link estimator of the routing protocol.

- LinkETX from the source node to its first hop parent, measured through the routing protocol's link estimator (i.e., 4-bit link estimation in CTP [7]).

- LinkRetx. The number of retransmissions from the source node to its first hop parent during packet delivery. In our testbed, the maximum allowable retransmissions for a link is 5. If the packet delivery fails after 5 retransmissions, a node would try to send the packet to an alternative parent in its parent set, as described in [49]. If there is no retries, the value of $LinkRetx$ is 0.

- Primary parent of the source node (i.e., the best parent node based on CTP).

- Path information. Every packet carries a 4 bytes path measurement encoded based on compressed sensing. Type I packet also records its path up to 6 hops.

All the information is used to reconstruct the full path from source node to sink after the packet is received in gateway, based on the algorithm described in [48].

Network Observation Period: The completeness of the data is essential for routing analysis since all the nodes in the network must be seen as a whole. Thus, we use the packets collected from 2017-08-10 to 2017-09-06 when the data is more complete compared to other periods for our network dynamics and benchmark analysis.

6.3 Network Dynamics

This section analyzes the network dynamics from three perspectives. First, link level characteristics gives the picture of the physical environment of the testbed. Second, topological characteristics shows the routing dynamics of the network as a whole during its long-term operation. Third, the temporal characteristics shows the evolution of the network dynamics over time.

6.3.1 Link Level Characteristics

For each observed link, its measurement data is a time series tuples extracted from the received packet where each tuple contains U_RSSI,D_RSSI,$LinkETX$, and $LinkRetx$. For instance, the U_RSSI series for link $A \rightarrow B$ is: $U\_RSSI(s_1)$, $U\_RSSI(s_2)$, ..., $U\_RSSI(s_n)$, where s is the set of packets that contain link $A \rightarrow B$. There are 902 observed links in total in our observation period.

Based on the collected data, two types of link packet reception ratios (PRRs) can be computed as shown in (6.1) and (6.2). Both PRR values are also time series. PRR_etx is the moving average derived from the result of routing protocol's link estimation, whereas PRR_retx is the actual link PRR during data packet delivery.

$$PRR\_etx = \frac{1}{linkETX}, \tag{6.1}$$

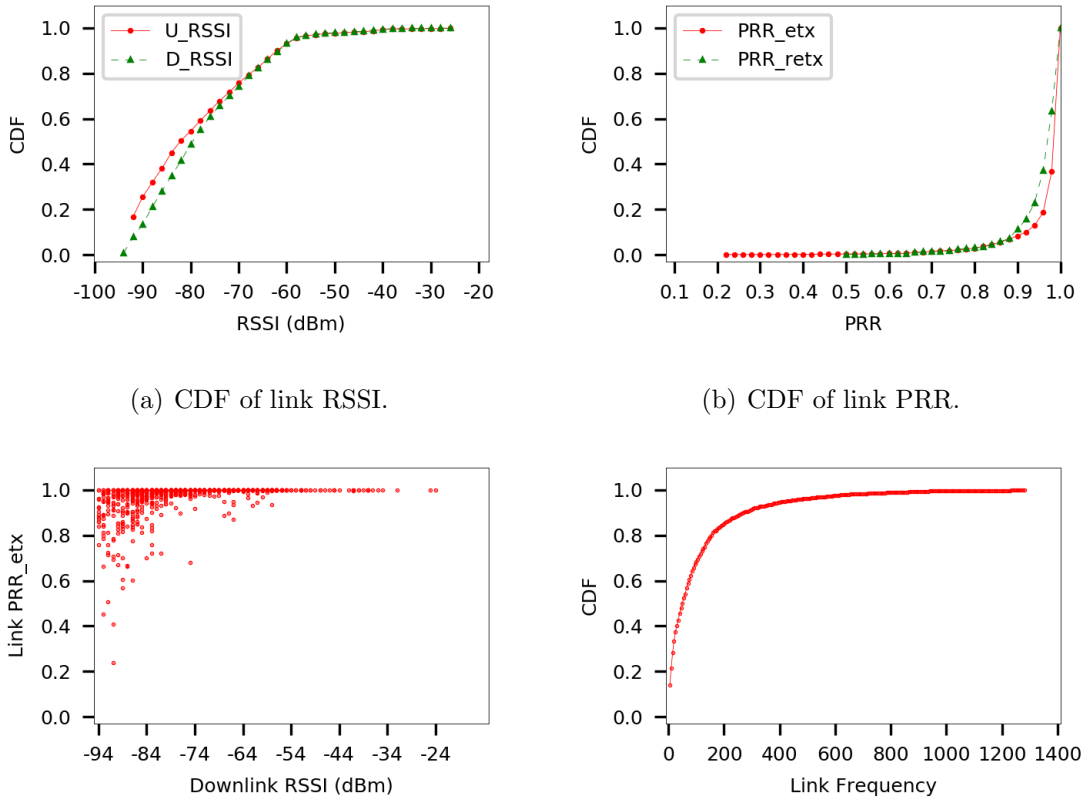
$$PRR\_retx = \frac{1}{linkRetx + 1}. \tag{6.2}$$

Overall Link Performance

The overall link performance in terms of the observed link RSSI and link PRR is presented in Figure 6.1. The value of a link is the average of its measurement time series. As we can see from Figure 6.1(a), majority of the links have RSSI in the range from -94 dBm to -60 dBm, but a few links have RSSI higher than -30 dBm. An apparent gap is observed between U_RSSI and D_RSSI, indicating an systemic link asymmetry exists in the testbed. As we will show later, the link asymmetry is mainly caused by the hardware heterogeneity of the node platforms.

Regarding link PRR, Figure 6.1(b) shows that the actual link PRR (i.e.,PRR_retx) is slightly worse than the PRR obtained from the link estimator. The 4-bit link estimator applies moving average to smooth the abrupt change of the link condition in order to reduce the fluctuation of the link conditions in the network. Thus, a delay exists for the link estimator to provide most accurate estimation. We argue that, given the observation in Figure 6.1(b), the coefficient in the moving average equation can be finer adjusted to better reflect the actual link conditions.

Previous works have studied the relationship between link RSSI and link PRR mostly in homogeneous networks [83, 84, 89]. We present our observation on the heterogeneous network in Figure 6.1(c) using the values D_RSSI (measured by link estimator) and PRR_etx (as in [84]). As the figure shows, the heterogeneity of the network results in a grey region, where RSSI values can have intermediate PRRs [86], of about 50 dBm which is significantly larger than 6 dBm reported in [86] and 11 dBm reported in [84]. This result demonstrates that the link characteristics in heterogeneous networks may considerably differ from that of homogeneous networks. Thus, those WSN algorithms (e.g., localization algorithms) that rely on RSSI measurements may need to be reconsidered when applying to heterogeneous networks.



(a) CDF of link RSSI.

(b) CDF of link PRR.

(c) *D_RSSI* v.s. *PRR_etx*.

(d) CDF of link frequency.

Figure 6.1. Overall link performance.

We also investigate the frequency of the links in the dataset. As shown in Figure 6.1(d), many links are short lived. For instance, about 68% of the links are used for less than 100 times, and about 85% of the links have been used for less than 200 times. The result indicates that the network has widely explored the usable links for data transmission and achieved high degree of network dynamics. Note that the detailed statistics of a link will be collected only when it is the first hop along a packet path of data collection.

Table 6.1.
Mote Radio Configurations

Mote Platform	Radio Chip	Antenna
MicaZ	CC2420, Max. 0 dBm	RSMA, gain 4.9 dBi
TelosB	CC2420, Max. 0 dBm	SMA, gain 5 dBi
IRIS	RF230, Max. 3 dBm	RSMA, gain 4.9 dBi

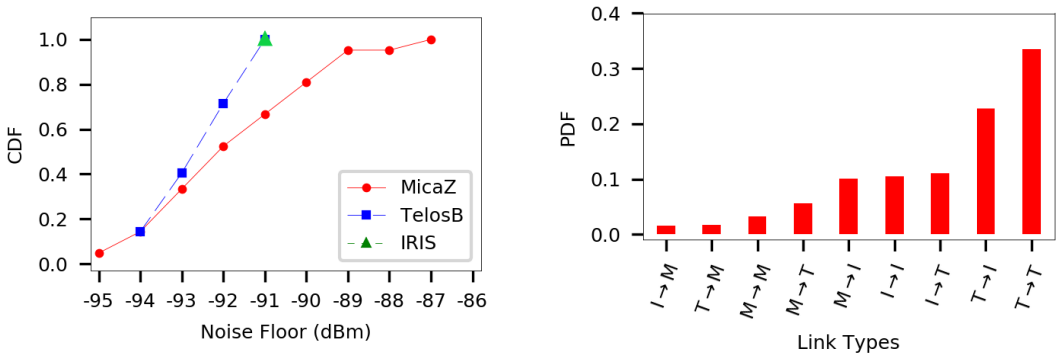
Impact of Mote Heterogeneity

In the section above we have observed a significantly different link RSSI and PRR behaviors from the previous reports for homogeneous networks (e.g., [84,86]). In this section we investigate the impact of mote heterogeneity to the link behaviors in more detail.

The ASWP WSN testbed contains three types of motes (i.e., MicaZ, TelosB, and IRIS), resulting in nine link types (e.g., $MicaZ \rightarrow IRIS$, or $M \rightarrow I$). The configurations of the transceiver of the mote platforms are summarized in Table 6.1.

To begin with, we show the noise floor observed by each node. Since the actual value is not transmitted with the packet, the noise floor at each node is approximated based on the minimum valid RSSI value sensed by this node [84], hence it maybe larger than the actual value. The result is shown in Figure 6.2(a). The significant differences are discovered between the mote platforms. IRIS motes are the most stable with a constant noise floor of -91 dBm (indicating an RSSI reading of 0 [94]). In contrast, the variance between MicaZ motes is the largest; TelosB motes sit in the middle.

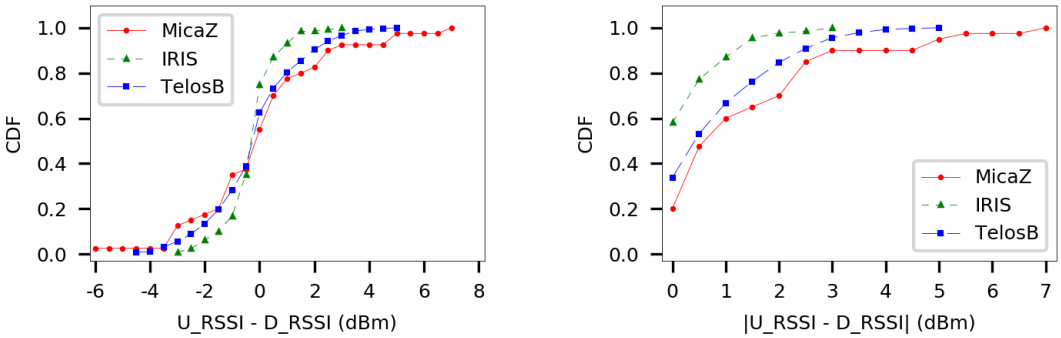
Figure 6.2(b) shows the percentage of each link types in our observed WSN test dynamics. About 76% of the links are within IRIS and TelosB motes, among which $TelosB \rightarrow TelosB$ ($T \rightarrow T$) links occupy 33%.



(a) CDF of noise floor.

(b) PDF of link types.

Figure 6.2. Distribution of noise floor and link types.

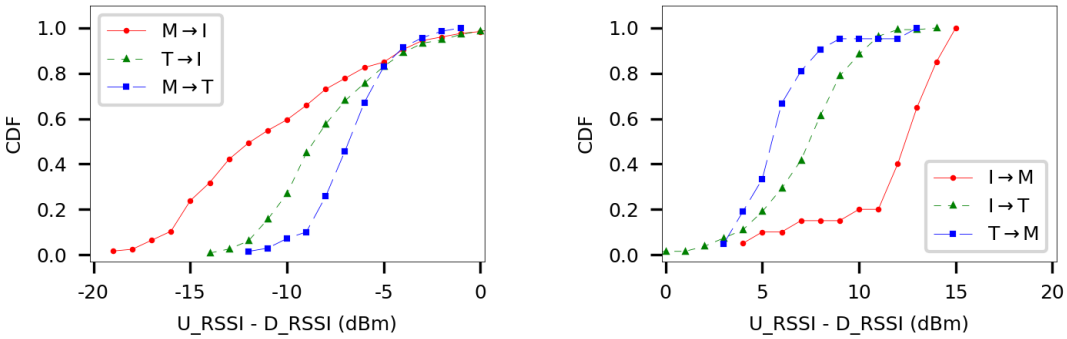


(a) Link asymmetry.

(b) Absolute Link asymmetry.

Figure 6.3. Link asymmetry based on RSSI for the links between the same types of nodes.

Next, we investigate the link asymmetry as a result of the mote heterogeneity. Figure 6.3, 6.4 presents the link asymmetry using the difference between U_RSSI and D_RSSI for different link types. For the links between the same type of motes, as shown in Figure 6.3(a), 6.3(b), IRIS motes have the least link asymmetry, MicaZ motes have the largest link asymmetry, and TelosB motes sit in the middle. If we define a link to be asymmetric if the difference between U_RSSI and D_RSSI is larger than 2 dBm, then 30% of the MicaZ links, 15% of the TelosB links, and



(a) Link asymmetry.

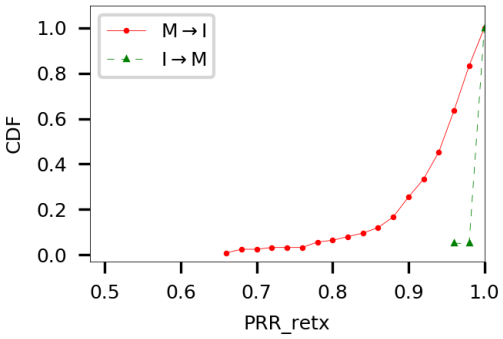
(b) Link asymmetry, reverse of Figure 6.4(a).

Figure 6.4. Link asymmetry based on RSSI for different link types.

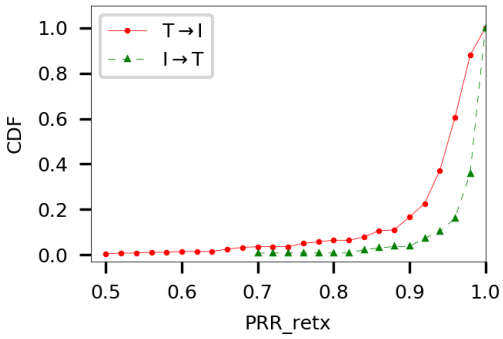
2.3% of the IRIS links are asymmetric. The links among different mote types are all asymmetric, as shown in Figure 6.4(a), 6.4(b). A constant bias is found between links among different mote types. For example, for links from MicaZ mote to IRIS mote, the U_RSSI (measured at IRIS mote) is always significantly smaller than the D_RSSI (measured at MicaZ mote). The reason is that the radio power of the IRIS mote is much stronger than that of the MicaZ mote. Hence, the signal strength from an IRIS mote to a MicaZ mote is much stronger than that from a MicaZ mote to an IRIS mote.

The link asymmetry is also observed based on link PRR (i.e., PRR_retx). As shown in Figure 6.5, the link PRR from the IRIS mote to other mote platforms is significantly better than the opposite direction due to IRIS mote's stronger radio signal. When the link PRR is less than 94%, the transmission from MicaZ mote to TelosB mote is better than the opposite direction.

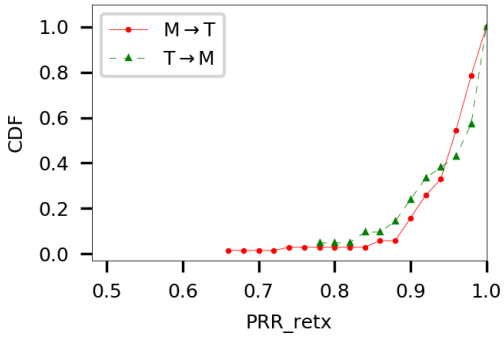
The results above demonstrate that the hardware heterogeneity is the dominating cause of the link asymmetry in the network.



(a) $MicaZ \leftrightarrow IRIS$.



(b) $TelosB \leftrightarrow IRIS$.



(c) $MicaZ \leftrightarrow TelosB$.

Figure 6.5. Link asymmetry based on link PRR_retx for different link types.

6.3.2 Topological Characteristics

We explore the path information to analyze the topological characteristics of the network during operation regarding the network as a whole. The network topology is extracted based on each data collection cycle with a length of 45 minutes. We also remove a cycle if it is identified as "bad", that is, if the cycle has too many missing paths.

To begin with, we present the node distribution on their average distance to the sink in the ASWP testbed, as shown in Figure 6.6. The nodes in the ASWP testbed are mostly distributed in distant hops from the sink due to the location restrictions (as indicated in Figure 4.12). IRIS nodes are the backbone of the network and are

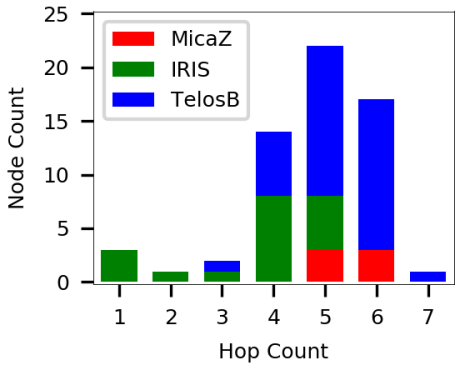


Figure 6.6. Node distribution on their average distance to the sink.

located at closer locations to the sink. TelosB nodes spread at a large portion of the network, whereas MicaZ nodes are at the most remote area.

The network density is estimated using the node degree based on the collected path information. The node degree is collected from each *child* $\leftrightarrow$ *parent* pairs. We also evaluate the possible parents a node can have based on the out-degree of the node. Note that the collected statistics is an approximation of the real value, which may be larger. However, the collected data presents the actual "useful" degree. Figure 6.7 shows the result. About 60% of the nodes have a degree larger than 20, indicating a dense node placement. Also, from the out-degree we observe that about 60% of the nodes has more than 15 possible parents for data packet forwarding. The result indicates that a node can have many alternative paths to reach the sink, hence a routing protocol should take this advantage to improve network performance.

The *de facto* standard routing protocols in WSNs are based on single path routing, such as CTP and RPL, where each node forwards the data packets to the current best/primary parent only. In contrast, our testbed relies on the CTP+EER routing protocol where each node actively explores a parent set for packet forwarding instead of a single parent. Thus, CTP+EER results in a more dynamic and load balanced network than that of CTP and RPL.

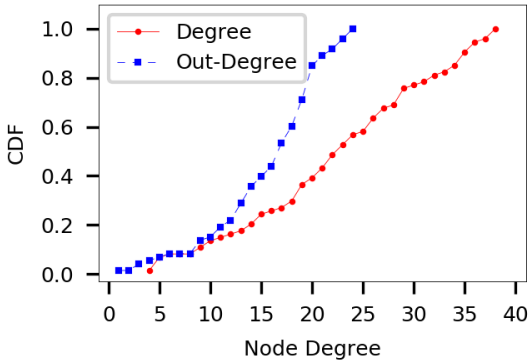


Figure 6.7. CDF of node degrees.

To illustrate the topological characteristics of the testbed, we also include the performance of CTP in comparison in order to provide a concrete understanding of the network dynamics. The statistics of CTP is inferred based on the primary parent. For each collection cycle, two topologies are generated for CTP+EER and CTP, respectively. For CTP+EER, the topology is based on the actual routing paths. For CTP, the topology and packet paths are inferred based on the primary parent, for instance, each packet in the CTP topology follows the path through the primary parent of each node. Note that the comparison is only meant to provide a concrete interpretation of the network dynamics in the ASWP testbed, instead of showing one protocol outperforms the other.

The current configuration of CTP+EER in the ASWP testbed allows each node to actively explore a maximum of 5 valid parents for packet forwarding at each moment. This is in contrast with CTP where a node usually triggers a reroute in order to change a parent. Figure 6.8 illustrates the spatial distribution of the average parent set size of the nodes in the ASWP testbed. A snapshot of the network topology is also included. Due to the unique shape of the testbed, nodes near the sink (i.e, first three hops) have very limited number of possible parents compared to the nodes in the main body of the network. The network is much denser at the remote areas, hence each node has more potential parents for packet forwarding.

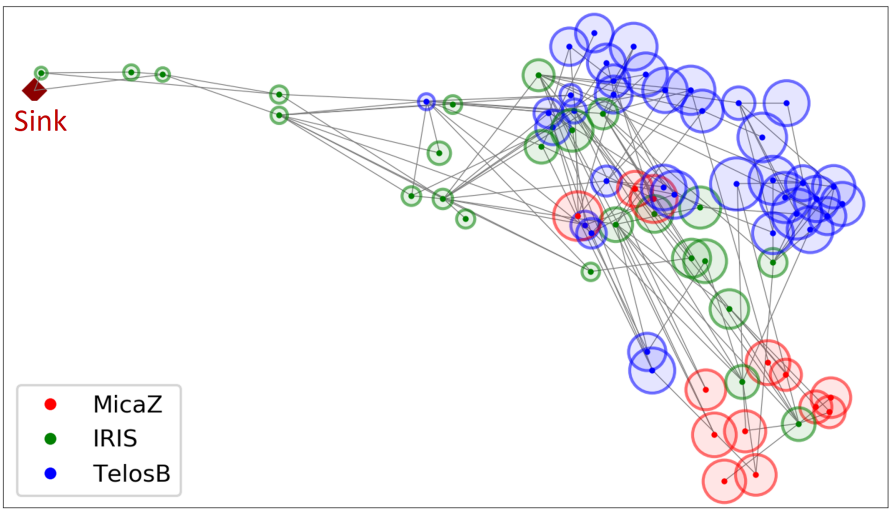


Figure 6.8. Spatial distribution of the average parent set size of the nodes at the ASWP testbed. The larger the circle, the larger the parent set.

To measure the network dynamics of the ASWP testbed and have a concrete comparison between CTP+EER and CTP, we define the following metrics for each topology to shows its characteristics:

- The number of links per cycle indicates the link/path diversity of each topology.
- Link entropy can be used to demonstrate the extent of routing dynamics.
- Node entropy can be used to show the load balance of the network.

The entropy of each cycle is computed as

$$S = -\sum_{i} p_i \ln p_i. \tag{6.3}$$

For link entropy, p_i is the percentage of packet forwards using this link to the total packet forwards in the cycle. For node entropy, p_i is the percentage of packet forwards using this node to the total packet forwards in the cycle. Both entropies are highly related to each other.

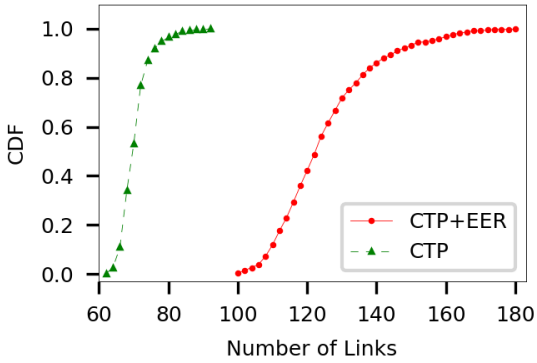
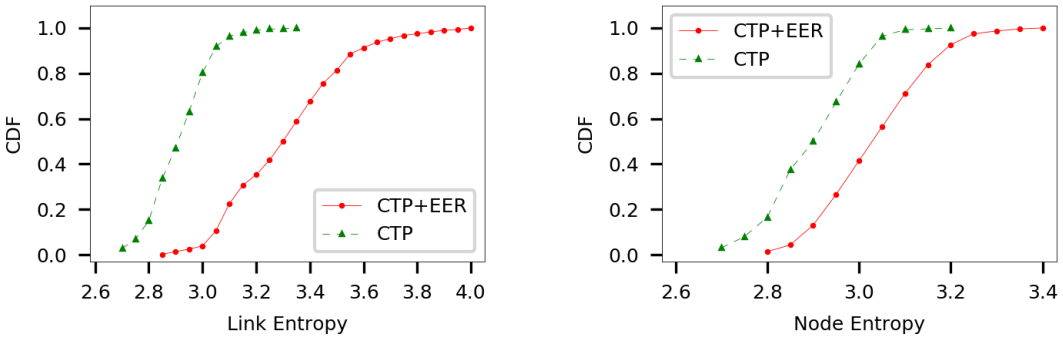


Figure 6.9. CDF of the number of links per cycle.



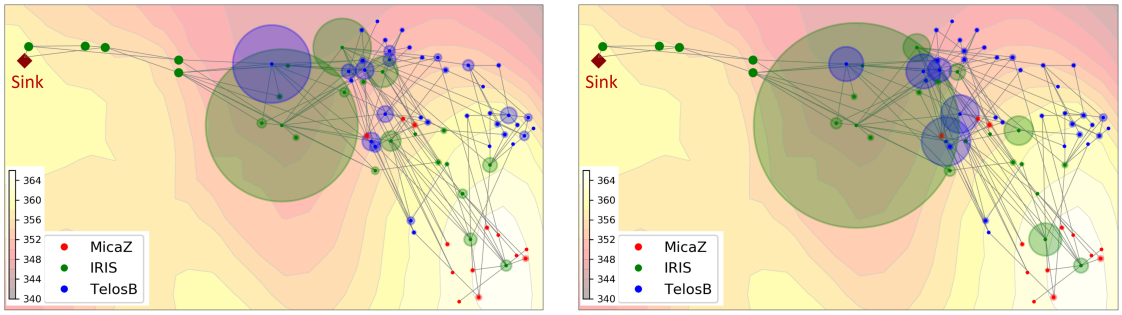
(a) Link entropy.

(b) Node entropy.

Figure 6.10. CDF of the link entropy and the node entropy.

Figure 6.9 gives the statistics of the number of links per cycle. Our testbed (using CTP+EER) results in much more routing dynamics compared to those using CTP since CTP+EER actively explores network dynamics even when the environment is stable in order to achieve load balance. The effect is demonstrated in Figure 6.10, where both entropies of CTP+EER are significantly higher than that of CTP.

As an illustration, we present in Figure 6.11 the number of forwarded packets at each node in the ASWP testbed during an example collection cycle. Due to the physical restriction of the node locations, the nodes in the first two hops simply forwards all the packets in the network, and are represented in solid dot. For other



(a) CTP+EER. (b) CTP.

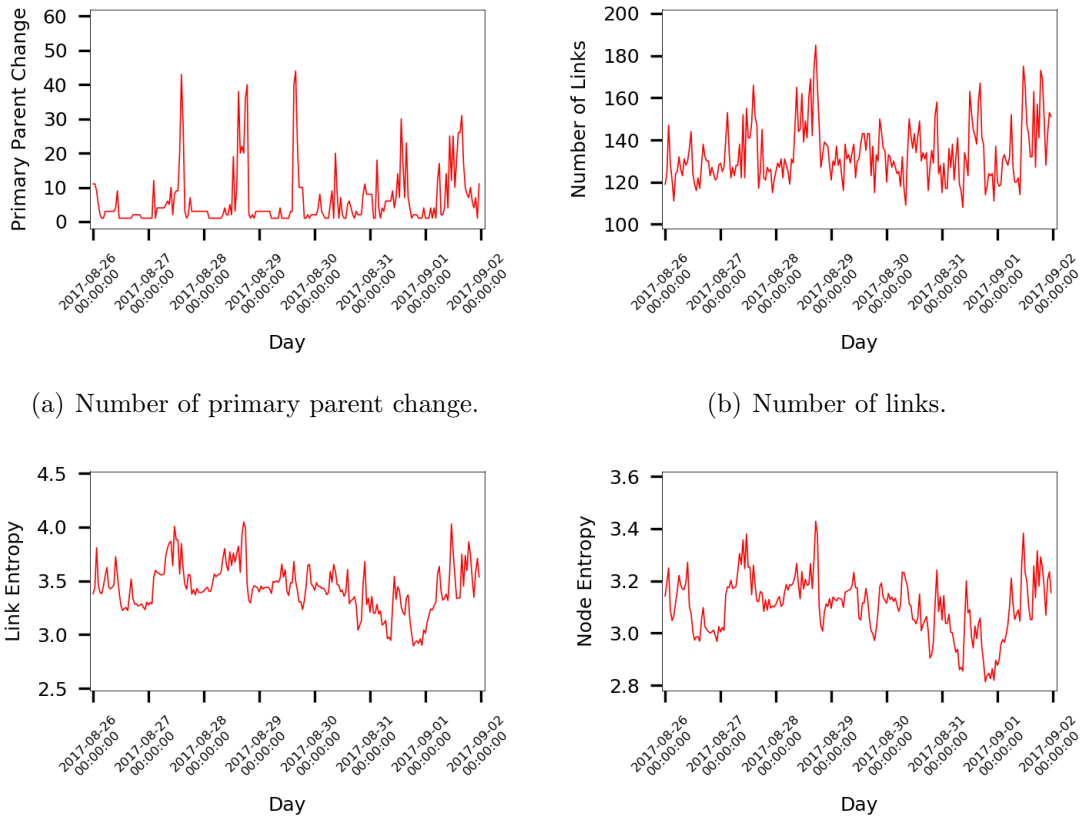
Figure 6.11. Spatial distribution of the number of forwarded packets for each node at the ASWP testbed. The larger the circle, the more concentration of the traffic. Sink's two-hop neighbors have forwarded all the traffic and are shown in solid green dots.

nodes, as we can see, the CTP network has larger circles indicating that the traffic is more concentrate. In comparison, the traffic in the CTP+EER network spread more evenly than that of CTP.

The topological characteristics presented above indicates that our testbed is very dynamic and the network traffic is less concentrate, hence achieves better network balance.

6.3.3 Temporal Characteristics

As the physical environment is constantly changing, it is worthwhile to examine the temporal characteristics of the network over time. Figure 6.12 presents the temporal observations on a per cycle basis within a week, including the number of primary parent changes, the number of links, the link entropy, and the node entropy. As we can see from Figure 6.12(a), a clear 24-hour periodic pattern is observed for the number of parent changes, where the network is much more dynamic in the day time than the night time. Similar patterns are also observed for the other metrics, though not as clear as the parent change. In general, the peak values usually occur



(a) Number of primary parent change.

(b) Number of links.

(c) Link entropy.

(d) Node entropy.

Figure 6.12. Temporal characteristics.

in the day time, and the dip values usually occur at the night time. The observation matches those reported in [84] and [92], indicating the environmental changes in day and night may affect the network performance.

6.4 Benchmark Data Suite

The original collected data is usually incomplete and has missing paths due to the dynamics of the physical environment. This section presents our method to generate the benchmark data suite and some basic characteristics of the data.

6.4.1 Data Generation

The data generation process is shown in Figure 6.13. The data is processed on a per cycle basis. The raw data is incomplete and contains error readings. In order to provide the community a readily usable dataset, we performed data cleanup and devised our approach to fill the missing paths. After a basic cleanup, the bad cycles are identified and removed from the dataset. We define a collection cycle to be "bad" if the number of missing paths is more than 11%. In total 90 cycles were removed since they were "bad", remaining 746 good cycles as our cleaned original data. Then, two datasets are generated after applying our filling missing paths method for each cycle. One dataset keeps the loopy packets intact, the other has the loopy packets processed to remove the loop in the path. For example, after processing, a loopy path (A, B, C, B, C, D) becomes (A, B, C, D), that is, the loop (B, C, B, C) is resolved. In the end, three datasets are provided as our benchmark date suite: the cleaned original data, the benchmark with loopy packets, and the benchmark without loopy packets.

Each data entry is a complete packet organized in the order of its cycle index. Each packet mainly contains the following data fields: *timestamp, cycle index, mote type, link* U_RSSI*, link* B_RSSI*, link ETX, link Retx, primary parent*, and *full path* from source node to the sink. Also, a $flag$ field is used to identify whether it is an original packet or a supplement packet to fill a missing path. The details of the data schema is described in the Appendix.

6.4.2 Fill Missing Paths

One of the main feature of our benchmark data suite is the full path of each packet that provides complete topological information. If a node's packet hences its path is missing in a cycle, then the cycle's topology is incomplete. Thus, our goal is to find the best path for a missing node that complements the topology. Since the principle of CTP+EER lead to the maximization of the network entropy, the best supplement path is the one that can maximize the network entropy when adding to the topology.

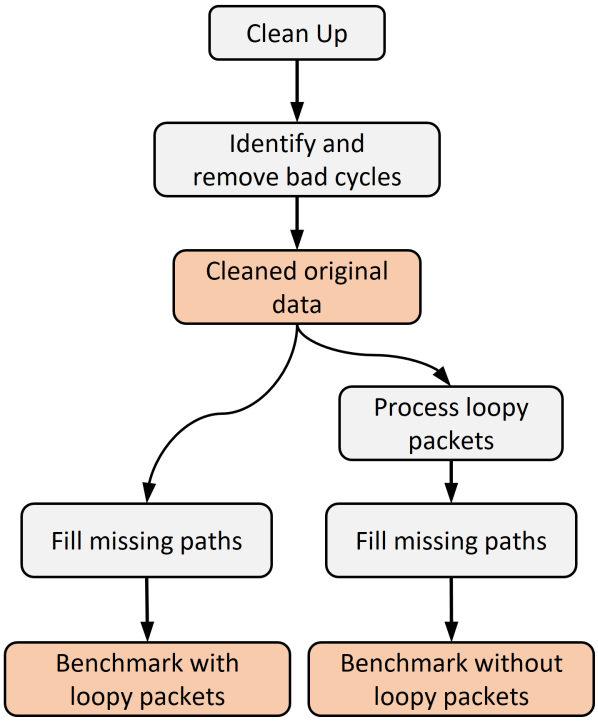


Figure 6.13. Benchmark data generation process.

A supplement packet for a missing path is examined in three time scales. (1) The same time period in the previous day is considered to be the best time interval to search for the supplement packet. (2) If a supplement packet is not found in the previous day, then the search range is extended to 2 days before and after the current time. (3) If the supplement packet is still not found in step (2), then the search range is extended for a month. Algorithm 3 describes how to find the best supplement packet in each period.

For each search interval, a set of candidate packets may be found for the missing path (line 4). A candidate packet must be verified to check whether it follows current $(child, parent)$ relationships in current topology (line 6). If a candidate packet can fit in current topology, the link entropy is computed and the best packet is selected as the one maximize the link entropy (line 7∼9). It is possible that none candidate

Algorithm 3: Find the Best Supplement Packet

Notations:

best_pkt: the best packet to for the missing path.

max_entropy: the current maximum link entropy after adding the candidate path to the topology.

topo: current topology.

interval: the time interval to search for a supplement packet for the missing path.

findCandidatePackets(target, interval): find all the packets of the missing *target* in the given time interval.

validatePacket(pkt, topo): check whether the given *pkt* can fit in the given topology.

linkEntropy(pkt, topo): compute the link entropy by adding the given *pkt* to current topology.

```
1  Function findBestPacket(target, interval)
2      best_pkt = null
3      max_entropy = 0
4      pkt_cands = findCandidatePackets(target, interval)
5      for (pkt in pkt_cands) do
6          if validPacket(pkt, topo) then
7              if linkEntropy(pkt, topo) > max_entropy then
8                  best_pkt = pkt
9                  max_entropy = linkEntropy(pkt, topo)
10             end
11         end
12     end
13     return best_pkt
```

packets are found or all the candidate packets do not fit to current topology, in this case, no packet can be used to complement the missing path.

Note that since the original data is provided, the users can apply their own methods to fill the missing paths based on their own criterion.

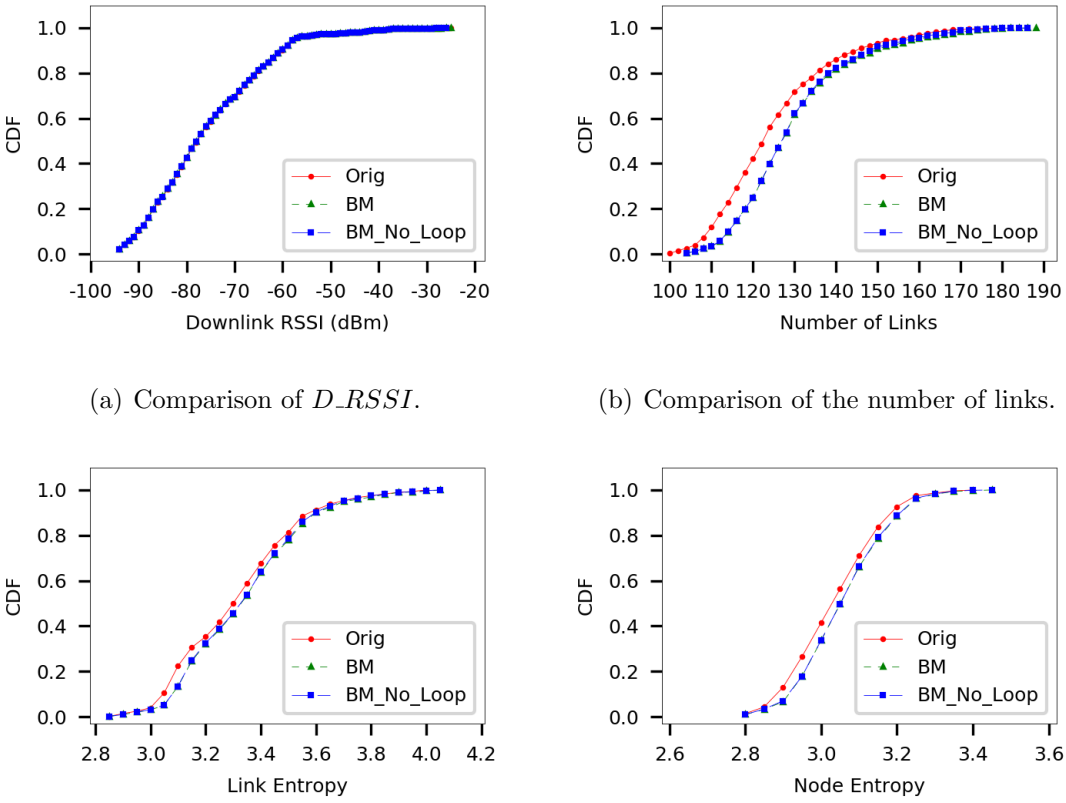
6.4.3 Data Characteristics

The overall statistics of the datasets is given in Table 6.2. About 2.1% packets in the benchmark (BM) datasets are supplements for missing paths per cycle. In total, 638 cycles has at least one missing path, and each cycle has about 3 missing paths on average. Since the supplement packets can be searched from the time period not covered by the dataset, more links can be added to the BM datasets than the original data. The loopy packets occupy about 0.3% of the total packets and spread in 95 cycles.

Table 6.2.
Overall Statistics of the Benchmark Datasets

DataSet	**Original**	**BM**	**BM_No_Loop**
Nodes	73 + sink		
Cycles	746		
Packets	107715	110051	110052
Filled packets	0	2336 (2.1%)	2337 (2.1%)
Cycles with filled packets	0	638	638
Loopy packets	334 (0.3%)	334 (0.3%)	–
Cycles with loopy packets	95	95	0
Links	902	906	906

By filling the missing paths, we concern about the network characteristics of the BM datasets compared to the original data. As we can see from Figure 6.14(a), the supplement packets do not have statistical significant effects on the link level behaviors of the datasets. Regarding the topological characteristics, our missing paths filling method aims to maximize the link entropy hence the network dynamics per cycle. The resulting statistics is shown in Figure 6.14(b), 6.14(c), and 6.14(d). As we can see, the BM and BM without loopy packets (BM_No_Loop) datasets have more links and larger entropies compared to the original dataset, demonstrating the effect of our method. Note that different missing paths filling method may results in different network characteristics.



(a) Comparison of D_RSSI.

(b) Comparison of the number of links.

(c) Comparison of link entropy.

(d) Comparison of node entropy.

Figure 6.14. Statistics of the original data, the benchmark data with loopy packets (BM), and the benchmark data without loopy packets (BM_No_Loop).

7 SUMMARY AND FUTURE WORK

7.1 Summary

In this dissertation, we address the energy efficient downstream communication problem in large scale, low power, and heterogeneous WSNs. We first address the downstream communication from the perspectives of small data dissemination and bulk data dissemination, then present an empirical analysis of a WSN deployment and devise a benchmark data suite.

By introducing opportunistic routing into the traditional source routing approach, we presented OSR, an Opportunistic Source Routing approach and protocol which provides reliable and scalable downward actuation in large-scale WSN systems. The unique opportunistic nature of OSR effectively addresses the fundamental issues of the drastic wireless link dynamics in noisy and resource-constrained WSNs. OSR only stores the direct child set at each intermediate node rather than the entire subtree of descendants as other address-based routing protocols. As a result, OSR has small memory overhead and achieves great scalability while maintaining good performance. The results on our simulations and real-world WSN testbed experiments demonstrate the merits of OSR. OSR significantly outperforms RPL storing mode and non-storing mode on two most widely used implementations. On the other hand, while OSR achieves desirable and comparable packet delivery rate as the flooding based Drip, it has much lower duty cycle in comparison with Drip.

We propose MobileDeluge, a novel mobile reprogramming tool which is able to reprogram heterogeneous WSNs regularly operating over low-power links. We have evaluated the performance of MobileDeluge through laboratory experiments and real-world outdoor WSN testbed reprogramming. Results show that MobileDeluge has efficiently addressed the reprogramming challenges of heterogeneous WSNs and WSNs

over low power links at the same time, making it very suitable for long-term outdoor WSN deployments where on-site maintenance is usually needed. The design of MobileDeluge also illustrates a general approach for building a mobile code dissemination tool based on some existing code dissemination protocol, such as Deluge, which in principle can be applied to other existing code dissemination protocols as well. If outdoor WSN deployments are not accessible by the maintenance team, a new code dissemination protocol with the fixed control/sink node would need to be developed for heterogeneous WSNs over low-power links.

We present an empirical analysis of the network dynamics for an outdoor heterogeneous WSN deployment, including the link level characteristics, topological characteristics, and temporal characteristics. We devise a benchmark data suite based on the data collected from the deployment. The main features of the benchmark includes the link information between heterogeneous hardware platforms and the full topological information. Analysis results show that asymmetric links are the majority in heterogeneous networks and the main cause is the heterogeneity in radio hardware. For topological analysis, our testbed operates on CTP+EER which actively explores alternative paths to the sink in order to achieve load balancing, hence the network topology is much dynamic than those use CTP. Since the raw data is incomplete and dirty, we present our method to clean up the data and to fill the missing paths. Our missing paths filling method aims to maximize the network dynamics to match the principle of CTP+EER. As a result, three datasets are generated to form the benchmark date suite: the cleaned original data with missing values, the benchmark data with supplement packets and loopy packets, and the benchmark data with supplement packets but without loopy packets. While the three datasets have no significant statistical difference on their link level behaviors, the BM datasets with supplement packets are more dynamic than the original dataset. The result demonstrates the effectiveness of our missing paths filling method.

7.2 Future Work

For downward unicast routing, the future work includes to extend OSR for WSN downward multicast routing, and to apply/integrate OSR with RPL non-storing mode. It is straightforward to use Bloom filter to encoded multiple targets. Multiple paths can be included in one path Bloom filter as well. By integrating with RPL, OSR would obtain the capability to work in IP based networks which can largely extend its usage. We believe OSR provides a significant and practical solution to wireless actuation for large-scale and resource-constrained WSN deployments.

MobileDeluge provides the concept of mobile control/query system, which enables efficient on field diagnosis. One of the future direction is to design a general framework for out-door heterogeneous WSN deployments which includes mote reprogramming, data query, command system, and network diagnosis, to provide as much facility as possible to reduce the laborious work on the field.

Analyzing network dynamics requires more complete topological information than pure link level analysis. In the benchmark analysis of the ASWP WSN testbed, the analyzed data period is less than a month due to the difficulty to gather enough complete data. One of the main reason is that the battery conditions on the sensor nodes vary in a large extent during long-term testbed maintainance. Thus, nodes die at random time instance, breaking the intactness of the data. Recently we found that the lithium batteries are good candidate in long-term WSN deployment due to its high capacity and low self discharge. Thus, one of the future direction is to install the lithium batteries to the sensor nodes and obtain complete data in a much longer time. Then, a more comprehensice analysis of the network dynamic can be conducted, for instance, based on different season of the year. A better benchmark data suite can be generated as well.

REFERENCES

REFERENCES

[1] Miguel Navarro, Tyler W Davis, German Villalba, Yimei Li, Xiaoyang Zhong, Newlyn Erratt, Xu Liang, and Yao Liang. Towards long-term multi-hop WSN deployments for environmental monitoring: An experimental network evaluation. *Journal of Sensor and Actuator Networks*, 3(4):297–330, 2014.

[2] Adam B Noel, Abderrazak Abdaoui, Tarek Elfouly, Mohamed Hossam Ahmed, Ahmed Badawy, and Mohamed S Shehata. Structural health monitoring using wireless sensor networks: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 19(3):1403–1423, 2017.

[3] Nagender Kumar Suryadevara, Subhas Chandra Mukhopadhyay, Sean Dieter Tebje Kelly, and Satinder Pal Singh Gill. WSN-based smart sensors and actuator for power management in intelligent buildings. *IEEE/ASME Transactions on Mechatronics*, 20(2):564–571, 2015.

[4] X. Mao, X. Miao, Y. He, X. Y. Li, and Y. Liu. CitySee: Urban CO_2 monitoring with sensors. In *Proceedings of the 2012 IEEE Conference on Computer Communications*, pages 1611–1619, March 2012.

[5] Mare Srbinovska, Cvetan Gavrovski, Vladimir Dimcev, Aleksandra Krkoleva, and Vesna Borozan. Environmental parameters monitoring in precision agriculture using wireless sensor networks. *Journal of Cleaner Production*, 88:297–307, 2015.

[6] Hairong Yan, Hongwei Huo, Youzhi Xu, and Mikael Gidlund. Wireless sensor network based e-health system-implementation and experimental results. *IEEE Transactions on Consumer Electronics*, 56(4):2288–2295, 2010.

[7] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, Maria Kazandjieva, David Moss, and Philip Levis. CTP: An efficient, robust, and reliable collection tree protocol for wireless sensor networks. *ACM Transactions on Sensor Networks*, 10(1):16, 2013.

[8] Tim Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, K. Pister, R. Struik, J. P. Vasseur, and R. Alexander. RPL: IPv6 routing protocol for low-power and lossy networks. *RFC6550*, 2012.

[9] Gilman Tolle and David Culler. Design of an application-cooperative management system for wireless sensor networks. In *Proceeedings of the 2nd European Workshop on Wireless Sensor Networks*, pages 121–132, Jan 2005.

[10] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks*, pages 73–84. IEEE, 2011.

[11] Jonathan W. Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 81–94, New York, NY, USA, 2004. ACM.

[12] Tian He, Sudha Krishnamurthy, Liqian Luo, Ting Yan, Lin Gu, Radu Stoleru, Gang Zhou, Qing Cao, Pascal Vicaire, John A Stankovic, et al. VigilNet: An integrated sensor network system for energy-efficient surveillance. *ACM Transactions on Sensor Networks*, 2(1):1–38, 2006.

[13] Sandip Bapat, Vinodkrishnan Kulathumani, and Anish Arora. Analyzing the yield of exscal, a large-scale wireless sensor network experiment. In *Proceedings of the 13th IEEE International Conference on Network Protocols*, pages 10–pp. IEEE, 2005.

[14] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, et al. A macroscope in the redwoods. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, pages 51–63. ACM, 2005.

[15] Guillermo Barrenetxea, François Ingelrest, Gunnar Schaefer, Martin Vetterli, Olivier Couach, and Marc Parlange. Sensorscope: Out-of-the-box environmental monitoring. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, pages 332–343. IEEE Computer Society, 2008.

[16] H. S. Kim, J. Ko, D. E. Culler, and J. Paek. Challenging the IPv6 routing protocol for low-power and lossy networks (RPL): A survey. *IEEE Communications Surveys Tutorials*, 19(4):2502–2525, Fourth quarter 2017.

[17] Hyung-Sin Kim, Myung-Sup Lee, Young-June Choi, Jeonggil Ko, and Saewoong Bahk. Reliable and energy-efficient downward packet delivery in asymmetric transmission power-based networks. *ACM Transactions on Sensor Networks*, 12(4):34:1–34:25, 2016.

[18] H. S. Kim, H. Im, M. S. Lee, J. Paek, and S. Bahk. A measurement study of TCP over RPL in low-power and lossy networks. *Journal of Communications and Networks*, 17(6):647–655, Dec 2015.

[19] Hyung-Sin Kim, Hosoo Cho, Hongchan Kim, and Saewoong Bahk. DT-RPL: Diverse bidirectional traffic delivery through RPL routing protocol in low power and lossy networks. *Computer Networks*, 126:150–161, 2017.

[20] O. Iova, P. Picco, T. Istomin, and C. Kiraly. RPL: The routing standard for the internet of things... or is it? *IEEE Communications Magazine*, 54(12):16–22, December 2016.

[21] Timofei Istomin, Csaba Kiraly, and Gian Pietro Picco. Is RPL ready for actuation? A comparative evaluation in a smart city scenario. In *European Conference on Wireless Sensor Networks*, pages 291–299. Springer, 2015.

[22] T. Clausen, U. Herberg, and M. Philipp. A critical evaluation of the IPv6 routing protocol for low power and lossy networks (RPL). In *Proceedings of the 7th International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 365–372, Oct 2011.

[23] C. Kiraly, T. Istomin, O. Iova, and G. P. Picco. D-RPL: Overcoming memory limitations in RPL point-to-multipoint routing. In *Proceedings of the 40th IEEE Conference on Local Computer Networks*, pages 157–160, Oct 2015.

[24] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. Orchestra: Robust mesh networks through autonomously scheduled TSCH. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 337–350, New York, NY, USA, 2015. ACM.

[25] Simon Duquennoy, Olaf Landsiedel, and Thiemo Voigt. Let the tree bloom: Scalable opportunistic routing with ORPL. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, pages 2:1–2:14, New York, NY, USA, 2013. ACM.

[26] A. Reinhardt, O. Morar, S. Santini, S. Zller, and R. Steinmetz. CBFR: Bloom filter routing with gradual forgetting for tree-structured wireless sensor networks with mobile nodes. In *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–9, June 2012.

[27] Kaisen Lin and Philip Levis. Data discovery and dissemination with DIP. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, pages 433–444, Washington, DC, USA, 2008. IEEE Computer Society.

[28] Thanh Dang, Nirupama Bulusu, Wu-Chi Feng, and Seungweon Park. DHV: A code consistency maintenance protocol for multi-hop wireless sensor networks. In *Proceedings of the 6th European Conference on Wireless Sensor Networks*, pages 327–342, Berlin, Heidelberg, 2009. Springer-Verlag.

[29] Zygmunt J Haas, Joseph Y Halpern, and Li Li. Gossip-based ad hoc routing. *IEEE/ACM Transactions on Networking*, 14(3):479–491, 2006.

[30] P. Kyasanur, R. R. Choudhury, and I. Gupta. Smart gossip: An adaptive gossip-based broadcasting service for sensor networks. In *Proceedings of the 3rd International Conference on Mobile Ad Hoc and Sensor Systems*, pages 91–100, Oct 2006.

[31] Mahesh Umamaheswaran Arumugam. Infuse: A TDMA based reprogramming service for sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 281–282, New York, NY, USA, 2004. ACM.

[32] Jaein Jeong, Sukun Kim, and Alan Broad. Mote in-nework programming user reference, 2003 (accessed March 28, 2018). `https://www.eol.ucar.edu/isf/facilities/isa/internal/TinyOSdoc/Xnp.pdf`.

[33] Thanos Stathopoulos, John Heidemann, and Deborah Estrin. A remote code update mechanism for wireless sensor networks. Technical report, UCLA, 2003.

[34] S. S. Kulkarni and Limin Wang. MNP: Multihop network reprogramming service for sensor networks. In *Proceedings of the 25th International Conference on Distributed Computing Systems*, pages 7–16, June 2005.

[35] David Moss and Philip Levis. BoX-MACs: Exploiting physical and link layer boundaries in low power networking. Technical report, Stanford University SING-08-00, 2008.

[36] *TinyOS Documentation Wiki*, 2013 (accessed by March 28, 2018). `http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyOS_Documentation_Wiki`.

[37] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC language: A holistic approach to networked embedded systems. *ACM SIGPLAN Notices*, 49(4):41–51, Jul 2014.

[38] Girts Strazdins, Atis Elsts, Krisjanis Nesenbergs, and Leo Selavo. Wireless sensor network operating system design rules based on real-world deployment survey. *Journal of Sensor and Actuator Networks*, 2(3):509–556, 2013.

[39] M. Amjad, M. Sharif, M. K. Afzal, and S. W. Kim. TinyOS-new trends, comparative views, and supported sensing applications: A review. *IEEE Sensors Journal*, 16(9):2865–2889, May 2016.

[40] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 126–137, New York, NY, USA, 2003. ACM.

[41] H. Lee, A. Cerpa, and P. Levis. Improving wireless simulation through noise modeling. In *Proceedings of the 6th International Symposium on Information Processing in Sensor Networks*, pages 21–30, April 2007.

[42] Newlyn Erratt and Yao Liang. Compressed data-stream protocol: An energy-efficient compressed data-stream protocol for wireless sensor networks. *IET Communications*, 5(18):2673–2683, 2011.

[43] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2004.

[44] David B Johnson, David A Maltz, Josh Broch, et al. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. *Ad Hoc Networking*, 5:139–172, 2001.

[45] *IEEE 802.15 WPAN Task Group 4 (TG4)*, 2018 (accessed by March 25, 2018). `http://www.ieee802.org/15/pub/TG4.html`.

[46] Yi Gao, Wei Dong, Chun Chen, Jiajun Bu, Gaoyang Guan, Xuefeng Zhang, and Xue Liu. Pathfinder: Robust path reconstruction in large scale sensor networks with lossy links. In *Proceedings of the 21st International Conference on Network Protocols*, pages 1–10. IEEE, 2013.

[47] Yao Liang and Rui Liu. Routing topology inference for wireless sensor networks. *ACM SIGCOMM Computer Communication Review*, 43(2):21–28, April 2013.

[48] Rui Liu, Yao Liang, and Xiaoyang Zhong. Monitoring routing topology in dynamic wireless sensor network systems. In *Proceedings of the 23rd International Conference on Network Protocols*, pages 406–416. IEEE, 2015.

[49] Miguel Navarro and Yao Liang. Efficient and balanced routing in energy-constrained wireless sensor networks for data collection. In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, pages 101–113, USA, 2016. Junction Publishing.

[50] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks*, pages 73–84. IEEE, 2011.

[51] Shuo Guo, Liang He, Yu Gu, Bo Jiang, and Tian He. Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links. *IEEE Transactions on Computers*, 63(11):2787–2802, 2014.

[52] Andreas Reinhardt and Christian Renner. RoCoCo: Receiver-initiated opportunistic data collection and command multicasting for WSNs. In *European Conference on Wireless Sensor Networks*, pages 218–233. Springer, 2015.

[53] Víctor Julián Roselló Gómez-Lobo, David Boyle, Jorge Portilla Berrueco, Brendan O'Flynn, and Teresa Riesgo Alcaide. Route-back delivery protocol for collection tree protocol-based applications. In *European Conference on Wireless Sensor Networks*, pages 9–12, 2013.

[54] Claude Castelluccia and Pars Mutaf. Hash-based dynamic source routing. In *International Conference on Research in Networking*, pages 1012–1023. Springer, 2004.

[55] M. Sarela, C. Esteve Rothenberg, T. Aura, A. Zahemszky, P. Nikander, and J. Ott. Forwarding anomalies in Bloom filter-based multicast. In *Proceedings of the 2011 IEEE Conference on Computer Communications*, pages 2399–2407, April 2011.

[56] J. Tapolcai, A. Gulyas, Z. Heszbergery, J. Biro, P. Babarczi, and D. Trossen. Stateless multi-stage dissemination of information: Source routing revisited. In *2012 IEEE Global Communications Conference*, pages 2797–2802, Dec 2012.

[57] O. Bergmann, C. Bormann, S. Gerdes, and Chen H. *Constrained-Cast: Source-routed multicast for RPL, draft-ietf-roll-ccast-01*, 2017 (accessed by March 25, 2018). `https://tools.ietf.org/html/draft-ietf-roll-ccast-01`.

[58] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz. Theory and practice of Bloom filters for distributed systems. *IEEE Communications Surveys Tutorials*, 14(1):131–155, 2012.

[59] Andrei Broder and Michael Mitzenmacher. Network applications of Bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[60] Thomas Wang. *Integer Hash Function*, 2007 (accessed by March 25, 2018). `http://web.archive.org/web/20071223173210/http://www.concentric.net/~Ttwang/tech/inthash.htm`.

[61] Bob Jenkins. *4-byte Integer Hashing*, 2017 (accessed by March 25, 2018). `http://burtleburtle.net/bob/hash/integer.html`.

[62] Glenn Fowler, Landon Curt Noll, and Kiem-Phong Vo. *FNV Hash*, 2017 (accessed by March 25, 2018). `http://isthe.com/chongo/tech/comp/fnv/`.

[63] M.V. Ramakrishna and Justin Zobel. Performance in practice of string hashing functions. In *Proceedings of the 2nd International Conference on Database Systems for Advanced Applications*, pages 215–223, 1997.

[64] Manjunath Doddavenkatappa, Mun Choon Chan, and Akkihebbal L Ananda. Indriya: A low-cost, 3D wireless sensor network testbed. In *Proceedings of the 2011 International Conference on Testbeds and Research Infrastructures*, pages 302–316. Springer, 2011.

[65] Nicolas Tsiftes, Joakim Eriksson, and Adam Dunkels. Low-power wireless IPv6 routing with ContikiRPL. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 406–407, New York, NY, USA, 2010. ACM.

[66] *TinyRPL*, 2011 (accessed by March 25, 2018). `http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyRPL`.

[67] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*, pages 641–648, Nov 2006.

[68] Yimei Li and Yao Liang. Compressed sensing in multi-hop large-scale wireless sensor networks based on routing topology tomography. *arXiv preprint arXiv:1709.00604*, 2017.

[69] Xiaoyang Zhong, Miguel Navarro, German Villalba, Xu Liang, and Yao Liang. MobileDeluge: Mobile code dissemination for wireless sensor networks. In *Proceedings of the 11th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pages 363–370. IEEE, 2014.

[70] Newlyn Erratt and Yao Liang. The design and implementation of a general WSN gateway for data collection. In *Proceedings of the 2013 IEEE Wireless Communications and Networking Conference*, pages 4392–4397. IEEE, 2013.

[71] Miguel Navarro, Diviyansh Bhatnagar, and Yao Liang. An integrated network and data management system for heterogeneous WSNs. In *Proceedings of the 8th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pages 819–824. IEEE, 2011.

[72] L. Huang and S. Setia. CORD: Energy-efficient reliable bulk data dissemination in sensor networks. In *Proceedings of the 2008 IEEE Conference on Computer Communications*, April 2008.

[73] Rajesh Krishna Panta, Saurabh Bagchi, and Samuel P Midkiff. Zephyr: Efficient incremental reprogramming of sensor nodes using function call indirections and difference computation. In *Proceedings of USENIX Annual Technical Conference*, 2009.

[74] Jaein Jeong and David Culler. Scalable incremental network programming for multihop wireless sensors. *International Journal of Communications, Network and System Sciences*, 6(1):37, 2013.

[75] B. Mazumder and J. O. Hallstrom. An efficient code update solution for wireless sensor network reprogramming. In *Proceedings of the 2013 International Conference on Embedded Software*, pages 1–10, Sept 2013.

[76] Jiefan Qiu, Dong Li, Hailong Shi, and Li Cui. EasiLIR: Lightweight incremental reprogramming for sensor networks. *International Journal of Distributed Sensor Networks*, 10(1):120597, 2014.

[77] Yi Gao, Chun Chen, Xue Liu, Jiajun Bu, Wei Dong, and Xianghua Xu. Reprogramming over low power link layer in wireless sensor networks. In *Proceedings of the IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems*, pages 461–469. IEEE, 2013.

[78] German Villalba, Fernando Plaza, Xiaoyang Zhong, Tyler W Davis, Miguel Navarro, Yimei Li, Thomas A Slater, Yao Liang, and Xu Liang. A networked sensor system for the analysis of plot-scale hydrology. *Sensors*, 17(3):636, 2017.

[79] Lufeng Mo, Yuan He, Yunhao Liu, Jizhong Zhao, Shao-Jie Tang, Xiang-Yang Li, and Guojun Dai. Canopy closure estimates with GreenOrbs: Sustainable sensing in the forest. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 99–112, New York, NY, USA, 2009. ACM.

[80] Andrew Tridgell. *Efficient algorithms for sorting and synchronization.* PhD thesis, Australian National University Canberra, 1999.

[81] Daniel S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[82] A. Kouba, S. Chaudhry, O. Gaddour, R. Chaari, N. Al-Elaiwi, H. Al-Soli, and H. Boujelben. Z-Monitor: Monitoring and analyzing IEEE 802.15.4-based wireless sensor networks. In *Proceedings of the 36th IEEE Conference on Local Computer Networks*, pages 939–947, Oct 2011.

[83] Kannan Srinivasan, Prabal Dutta, Arsalan Tavakoli, and Philip Levis. An empirical study of low-power wireless. *ACM Transactions on Sensor Networks*, 6(2):16, 2010.

[84] J. Wang and W. Dong. Understanding the link-level behaviors of a large scale urban sensor network. In *Proceedings of the 12th International Conference on Mobile Ad-Hoc and Sensor Networks*, pages 15–22, Dec 2016.

[85] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 1–13, New York, NY, USA, 2003. ACM.

[86] Kannan Srinivasan, Prabal Dutta, Arsalan Tavakoli, and Philip Levis. Understanding the causes of packet delivery success and failure in dense wireless sensor networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, pages 419–420. ACM, 2006.

[87] Simon Duquennoy, Olaf Landsiedel, Carlo Alberto Boano, Marco Zimmerling, Jan Beutel, Mun Choon Chan, Omprakash Gnawali, Mobashir Mohammad, Luca Mottola, Lothar Thiele, et al. A benchmark for low-power wireless networking. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, pages 332–333. ACM, 2016.

[88] Carlo Alberto Boano, Simon Duquennoy, Anna Frster, Omprakash Gnawali, Romain Jacob, Hyung-Sin Kim, Olaf Landsiedel, Ramona Marfievici, Luca Mottola, Gian Pietro Picco, Xavier Vilajosana, Thomas Watteyne, and Marco Zimmerling. IoTBench: Towards a benchmark for low-power wireless networking. In *Proceeedings of the 1st Workshop on Benchmarking Cyber-Physical Networks and Systems*, 2018.

[89] Nouha Baccour, Anis Koubâa, Luca Mottola, Marco Antonio Zúñiga, Habib Youssef, Carlo Alberto Boano, and Mário Alves. Radio link quality estimation in wireless sensor networks: A survey. *ACM Transactions on Sensor Networks*, 8(4):34, 2012.

[90] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, pages 381–396. USENIX Association, 2006.

[91] Yunhao Liu, Yuan He, Mo Li, Jiliang Wang, Kebin Liu, and Xiangyang Li. Does wireless sensor network scale? A measurement study on GreenOrbs. *IEEE Transactions on Parallel and Distributed Systems*, 24(10):1983–1993, 2013.

[92] T. Zhu, W. Dong, Y. He, Q. Ma, L. Mo, and Y. Liu. Understanding routing dynamics in a large-scale wireless sensor network. In *Proceedings of the 10th International Conference on Mobile Ad-Hoc and Sensor Systems*, pages 574–582, Oct 2013.

[93] *Intel Lab Data*, 2004 (accessed by March 25, 2018). `http://db.csail.mit.edu/labdata/labdata.html`.

[94] *RF230 (AT86RF230)*, 2017 (accessed by March 25, 2018). `http://ww1.microchip.com/downloads/en/devicedoc/doc5131.pdf`.

APPENDIX

APPENDIX: BENCHMARK DATA FORMAT

Each entry in the benchmark dataset is a single packet including link information, path information, and some meta-data, as described in Table A.1.

The data file is a database dumped file using `pg_dump` of PostgreSQL. It can be easily restored to a database using the `psql` command:

```
psql -U <username> -d <dbname> -1 -f <filename>
```

Table A.1.
Data Format of the Benchmark Dataset

Data Fields	Example	Description
timestamp	"2017-08-19 09:59:09.081"	"The timestamp at the base station when the packet is received.
cycle_idx	300	The cycle number the packet belongs to. Packets should be grouped based on the cycle_idx.
packet_type	239	The type of the packet. i.e., Type I (239), II (205), or III (222).
flag	1	Indicate whether a packet is supplement or not.
source_id	61831	The ID of the node that originates the packet
motetype	2	The node platform of the source node. i.e., MicaZ (0), IRIS (1), or TelosB (2).
ctp_parent	60731	The primary parent of the source node.
eer_parent	61131	The first hop forwarder of this packet
eer_retx	0	The number of link retries from source node to eer_parent.
link_etx	10	The link ETX from the source node to eer_parent.
uplink_rssi	-62	Uplink RSSI sensed at the eer_parent upon the packet reception.
downlink_ rssi	-57	Downlink RSSI sensed at the source node during link estimation.
thl	8	Time has lived. The hop count from the source node to the sink.
rest_path	31301, ..., 1	The rest node IDs along the path from the source node to the sink.

VITA

VITA

Xiaoyang Zhong

Education

- PhD, Computer Science, Purdue University, West Lafayette, Indiana, USA
- BS, Electronic Engineering, University of Science and Technology of China, Hefei, Anhui, China

Research Interests

Internet of things, wireless sensor networks, and cyber-physical systems