# DEPENDABLE MESSAGING IN WIRELESS SENSOR NETWORKS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the

Graduate School of The Ohio State University

By

Hongwei Zhang, B.S., M.S.

* * * * *

The Ohio State University

2006

Dissertation Committee:

Anish Arora, Adviser

Prasun Sinha

Paul Sivilotti

Dong Xuan

Xiaodong Zhang

Approved by

________________________

Adviser

Graduate Program in Computer Science and Engineering

# ABSTRACT

Messaging is a basic service in sensornets. Yet the unique system and application properties of sensornets pose substantial challenges for the messaging design: Firstly, dynamic wireless links, constrained resources, and application diversity challenge the architecture and protocol design of sensornet messaging; Secondly, complex faults and large system scale introduce new challenges to the design of fault-tolerant protocols. The objective of this dissertation is to address the aforementioned challenges of sensornet messaging.

Despite the extensive effort in studying sensornet messaging, the lack of a basic understanding of its essential components has been an obstacle for reliable, efficient, and reusable messaging services in sensornets. To address this problem, one task of this dissertation is to identify the basic components of sensornet messaging and to study the related algorithmic design issues. More specifically, we propose the messaging architecture SMA that consists of three components: traffic-adaptive link estimation and routing (TLR), application-adaptive structuring (AST), and application-adaptive scheduling (ASC). TLR deals with dynamic wireless links as well as the impact of application traffic patterns on link dynamics; AST and ASC control the spatial and temporal flow of data packets to support application-specific in-network processing and QoS requirements. To provide an instance of the component TLR, we propose the routing protocol *Learn on the Fly* (LOF). LOF solves the problem of precisely estimating wireless link properties in the presence of varying network conditions. Having instantiated the foundational component TLR, we study component ASC from the perspective of in-network processing and QoS provisioning respectively. Taking packet packing as an example of in-network processing, we study the problem of scheduling packet transmissions to improve messaging efficiency (e.g., the degree of in-network aggregation). For the basic problem of reliable and real-time data transport in event-detection sensornets, we propose the protocol *Reliable Bursty Convergecast* (RBC) that innovates the window-less block acknowledgment scheme and the retransmission-aware differentiated contention control mechanism. Even though detailed study of AST is still a part of our future work, the architecture SMA provides a framework for sensornet messaging, and the study of components TLR and ASC provides the algorithmic references for instantiating SMA. This part of the dissertation work has also provided dependable messaging services for several real-world sensornet systems.

The second part of the dissertation addresses the challenges that complex faults and large system scale bring to the design of fault-tolerant protocols. For scalable dependability irrespective of fault complexity and system scale, we propose the concept of *local stabilization*. In a locally-stabilizing system, fault impact is locally contained around where the fault has occurred, and the time taken for the system to stabilize depends only on the size of the fault-perturbed region instead of the system size. For shortest path routing, a basic problem in messaging, we propose a locally-stabilizing protocol LSRP. Upon starting at an arbitrary state where the perturbation size is $p$, LSRP stabilizes to yield shortest path routes within $O(p)$ time, and the nodes affected by the perturbation are within $O(p)$ distance from the perturbed regions. The concept of local stabilization and the algorithmic approach of LSRP are generically applicable to other networking and distributed computing problems.

To my family.

## ACKNOWLEDGMENTS

To reach this stage in my Ph.D. study and this point in my life, I am indebted to many great people for their wisdom, support, and love.

I was fortunate to have met Dr. Anish Arora in early 2001. It is him who has given me the opportunity to conduct focused research in the past years, and it is him who has showed me the road to high quality work. While enjoying the freedom of independent thinking, I greatly appreciated his insightful advice on my research as well as life. I also greatly appreciate his patience in guiding me through the early stages of my Ph.D. study. I still remember that, when I was writing my first research paper in English, he went through and polished every sentence of the draft!

While working for the DARPA NEST research program, I was also fortunate enough to interact with and learn from many great scientists and researchers including Dr. Emre Ertin, Dr. Mohamed Gouda, Dr. Ted Herman, Dr. Sandeep Kulkarni, Dr. William Leal, Dr. Mikhail Nesterenko, Dr. Rajiv Ramnath, and Dr. Prasun Sinha. Their wisdom and humor have greatly enriched my Ph.D. experience.

Being a Ph.D. student in an active research department, I have also enjoyed and appreciated the advice and help from many other professors including Dr. Ten H. Lai, Dr. David Lee, Dr. Ming T. Liu, Dr. Paul Sivilotti, Dr. Dong Xuan, and Dr. Xiaodong Zhang. They have made my stay at The Ohio State University fun and fruitful.

Being a research intern at Motorola Labs in Summer 2005, I have greatly enjoyed working with my manager and friend Mr. Loren J. Rittle. I greatly appreciate his support during and after my internship at Motorola.

During my Ph.D. study, I have enjoyed working with many other fellow graduate students, both within and outside The Ohio State University. I have got the chance to work with Hui Cao, Young-ri Choi, Zhijun Liu, Vinayak Naik, and Lifeng Sang on shared projects or research problems, and it was a wonderful experience. I have also interacted extensively with many other graduate colleagues including Sandip Bapat, Murat Demirbas, Prabal Dutta, Vinodkrishnan Kulathumani, Santosh Kumar, Vineet Mittal, and Mukundan Sridhara. Their help and laughters have greatly enriched my life at The Ohio State University. I would also like to thank my many other friends for their continued support during my life and study.

I am indebted to my parents Wancai Zhang and Shunbi Wang for their unconditional love and support. I would also like to thank the rest of my family — who are too numerous to name individually — for their love and help. It is this family that has made me strong and courageous to be the one I am today.

Last, but certainly not least, I would like to thank my wife Yun Wang for her unconditional love, care, and laughter. Her love and support have been enabling me to focus on my research during the days and nights, the weekdays and weekends. Without her, I would not have been able to accomplish what I have achieved so far. Yun is my love, my inspiration, and my life.

## VITA

|  |  |
|---|---|
| January 13, 1975 | Born - Chongqing, China |
| 1997 | B.S. Computer Engineering, Chongqing University, China |
| 2000 | M.S. Computer Engineering, Chongqing University, China |
| 2000-2001 | Graduate Fellow, The Ohio State University |
| June - September 2005 | Research Intern, Motorola Labs, USA |
| 2001-present | Graduate Research Associate, The Ohio State University |

## PUBLICATIONS

### Research Publications

Anish Arora and Hongwei Zhang. "LSRP: Local Stabilization in Shortest Path Routing". *IEEE/ACM Transactions on Networking*, 14 (3):520-531, June, 2006.

Anish Arora, Prabal Dutta, Sandip Bapat, Vinod Kulathumani, Hongwei Zhang, Vinayak Naik, Vineet Mittal, Hui Cao, Murat Demirbas, Mohamed Gouda, Young-Ri Choi, Ted Herman, Sandeep Kulkarni, U. Arumugam, Mikhail Nesterenko, A. Vora, and M. Miyashita. "A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking". *Computer Networks (Elsevier)*, 46(5):605-634, December, 2004.

Hongwei Zhang and Anish Arora. "GS$^3$: Scalable Self-configuration and Self-healing in Wireless Sensor Networks". *Computer Networks (Elsevier)*, 43(4):459-480, November, 2003.

Hongwei Zhang, Anish Arora, and Prasun Sinha. "Learn on the Fly: Data-driven Link Estimation and Routing in Sensor Network Backbones". *25th IEEE International Conference on Computer Communications (INFOCOM)*, 2006.

Emre Ertin, Anish Arora, Rajiv Ramnath, Mikhail Nesterenko, Vinayak Naik, Sandip Bapat, Vinod Kulathumani, Mukundan Sridharan, Hongwei Zhang, and Hui Cao. "Kansei: A Testbed for Sensing at Scale". *5th IEEE/ACM International Conference on Information Processing in Sensor Networks Special Track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, 2006.

Vinayak Naik, Emre Ertin, Hongwei Zhang, and Anish Arora. "Wireless Testbed Bonsai". *2nd International Workshop on Wireless Network Measurement (WiNMee)*, 2006.

Hongwei Zhang, Anish Arora, Young-ri Choi, and Mohamed Gouda. "Reliable Bursty Convergecast in Wireless Sensor Networks". *6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2005.

Vinayak Naik, Anish Arora, Prasun Sinha, and Hongwei Zhang. "Sprinkler: A Reliable Data Dissemination Service for Wireless Embedded Devices". *26th IEEE Real-Time Systems Symposium (RTSS)*, 2005.

Anish Arora, Rajiv Ramnath, Emre Ertin, Prasun Sinha, Sandip Bapat, Vinayak Naik, Vinod Kulathumani, Hongwei Zhang, Hui Cao, Mukundan Sridhara, Santosh Kumar, Nick Seddon, Chris Anderson, Ted Herman, N. Trivedi, C. Zhang, Mohamed Gouda, Young-Ri Choi, Mikhail Nesterenko, R. Shah, Sandeep Kulkarni, M. Aramugam, L. Wang, David Culler, Prabal Dutta, Cory Sharp, Gille Tolle, Mike Grimmer, B. Ferriera, and Ken Parker. "ExScal: Elements of an Extreme Scale Wireless Sensor Networks". *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2005.

Hongwei Zhang and Anish Arora. "Brief Announcement: Continuous Containment and Local Stabilization in Path-vector Routing". *24th ACM Symposium on Principles of Distributed Computing (PODC)*, 2005.

Hongwei Zhang, Anish Arora, and Zhijun Liu. "A Stability-oriented Approach to Improving BGP Convergence". *23rd IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2004.

Anish Arora and Hongwei Zhang. "LSRP: Local Stabilization in Shortest Path Routing". *IEEE-IFIP International Conference on Dependable Systems and Networks (DSN)*, 2003.

Hongwei Zhang and Anish Arora. "GS$^3$: Scalable Self-configuration and Self-healing in Wireless Networks". *21st ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

Hongwei Zhang and Arjan Durresi. "Differentiated Multi-Layer Survivability in IP/WDM Networks". *8th IEEE-IFIP Network Operations and Management Symposium (NOMS)*, 2002.

## FIELDS OF STUDY

Major Field: Computer Science and Engineering

Studies in:

| | |
|---|---|
| Computer Networking | Prof. Anish Arora |
| | Prof. Prasun Sinha |
| | Prof. Dong Xuan |
| | Prof. Xiaodong Zhang |
| Software Systems | Prof. Paul Sivilotti |
| Computer Architecture | Prof. Mario Lauria |

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 Messaging: a challenge problem in sensornets

In wireless sensor networks, which we refer to as *sensornet*s hereafter, each node usually has limited capability in sensing, computing, and control. Therefore, nodes in sensornets coordinate with one another to perform tasks such as event detection and data collection [11]. Since nodes are spatially distributed, message passing is the basic enabler of node coordination in sensornets. This dissertation deals with the sensornet system services that are responsible for message passing; these services include consideration of routing and transport issues and we refer to them as the *messaging* services.

Despite the fact that messaging has been extensively studied for decades in traditional networks such as the Internet, the unique system and application properties of sensornets bring substantial challenges to the design of messaging services. Among others, the salient properties of sensornets and accordingly the challenges to sensornet messaging design are as follows:

- *Dynamic and potentially unreliable wireless links*: Unlike in wireline networks such as the Internet, wireless communication is subject to the impact of a variety of factors such as fading, multi-path, environmental noise, and co-channel interference. Thus wireless link properties (e.g., packet delivery rate) are dynamic and assume complex spatial and temporal patterns [102, 107]. For instance, Figure 1.1 shows how link

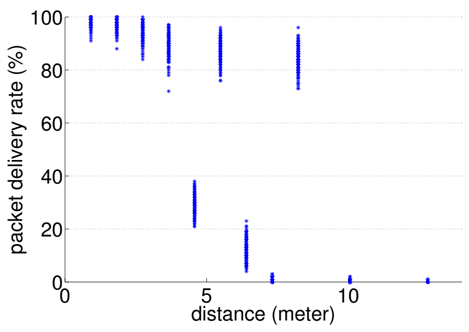


Figure 1.1: Scatter plot of broadcast packet delivery rate in Kansei [13]. There are 300 data points for each distance, with each data points representing the status of 100 broadcast transmissions. (Interested readers can find more detailed discussion of the experimentation environment in Section 3.2.)

reliability (i.e., packet delivery rate) changes with the transmitter-receiver distance in the sensornet testbed Kansei [13]. We see that link reliability tends to decrease as distance increases, but there exists a complex transition region (e.g., for distances between 2.74 meters and 12.8 meters) where link reliability may well increase as distance increases. For each specific link, its reliability also varies temporally. For instance, Figure 1.2 shows the time series of the packet delivery rates for a link corresponding to the 5.5-meter transmitter-receiver distance. We see that, even though the average link reliability is more than 85%, the temporal variation is significant, for instance, close to 20%. Similar phenonmena have been observed in other environments such as outdoor forested areas [106, 102] and urban environments [10].

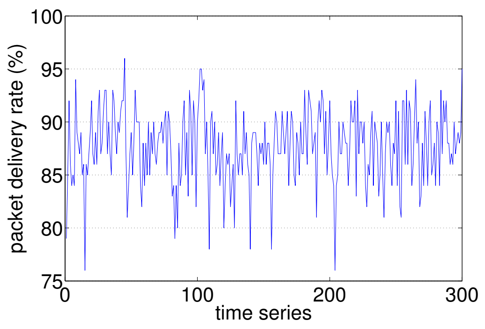


Figure 1.2: Time series of packet delivery rates of the link that corresponds to the 5.5-meter transmitter-receiver distance

Therefore, wireless communication poses the challenge of *guaranteeing reliable messaging in the presence of dynamic and potentially unreliable wireless links*.

- *Constrained resources*: Unlike devices in traditional networks, sensor nodes tend to be constrained in resources such as memory, CPU processing power, wireless channel bandwidth, and energy supply. For instance, the Tmote Sky sensor node [6] only has 10KB of RAM, 48KB of ROM, a 8MHz CPU, a radio of bandwidth 250Kbps, and two AA batteries as the energy supply. The resource constraints not only require network protocols to be of light-weight, they make it desirable is to break the traditional end-to-end network architecture [26] and to process information in the network, especially from the perspective of energy efficiency due to the high energy consumption in wireless communications. Messaging determines to a great extent the spatial and temporal flow of network traffic, thus it plays a significant role in affecting the degree of in-network processing achievable and thus the energy efficiency in sensornets.

Therefore, resource constraints challenge sensornet messaging *not only in terms of light-weight protocol design but also in terms of breaking the traditional network architecture and facilitating in-network processing*.

- *Application diversity*: With their unique capabilities in observing and controlling the physical world, sensornets have a broad range of potential applications in science (e.g., ecology and seismology), engineering (e.g., industrial control and precision agriculture), and our daily life (e.g., traffic control and health care). Due to the broad application domains, sensornet systems tend to differ in many ways including their traffic patterns and quality of service (QoS) requirements. For instance, in data-collection systems such as those for ecological study, application data are usually generated periodically, and the applications can tolerate certain degree of loss and delay in data delivery; yet in emergency-detection systems such as those for industrial control, data are generated only when rare and emergent events occur, but the data need to be delivered reliably and in real time. The implications of application diversity for messaging include:

  - Application traffic affects wireless link properties due to interference among simultaneous transmissions. This impact can vary significantly across diverse traffic patterns [102].
  - Different requirements of QoS and in-network processing pose different constraints on the spatial and temporal flow of application data [103].
  - Messaging services that are custom-designed for one application can be unsuitable for another, as is evidenced by a study of Zhang et al [4].

  - It is desirable that message services accommodate diverse QoS and in-network processing requirements.

  Therefore, diversity in applications poses the challenge of *designing messaging services that self-adapt to application properties*.

- *Complex faults and large system scale*: During the past years of field deployments of sensornet systems, we have observed a variety of faults, from the simple ones such as node or link failure to more complex ones such as state corruption and system signal loss [19]. We have also seen that sensornet systems tend to be of large scale (e.g., in terms of the number of nodes deployed) given the limited capability of each individual node. As network scales up, the overall probability of fault occurrence increases. Moreover, faults occurring at a small region can propagate unboundedly and affect a lot of nodes in the network, thus degenerating the stability and availability of the system.

  Therefore, the challenge here is *how to guarantee network dependability irrespective of system scale and fault complexity*.

**In summary, the basic challenges of sensornet messaging are twofold:**

- **Firstly, how to provide reliable and efficient messaging despite complex dynamics of wireless communication, constrained resources, and application diversity?**
- **Secondly, how to guarantee scalable dependability irrespective of system scale and fault complexity?**

This dissertation answers questions related to the above two problems.

## 1.2 Contributions of the dissertation

For reliable and efficient messaging despite dynamic wireless links, constrained resources, and application diversity, the dissertation addresses the architectural and algorithmic issues as follows:

- **[Architecture.]** To build the framework for reliable, efficient, and reusable messaging in sensornets, we propose the *Sensornet Messaging Architecture* (SMA) [103] which decomposes the messaging task via two levels of abstraction:
  - At the lower level, we identify the component *traffic-adaptive link estimation and routing* (TLR) that is responsible for precisely estimating wireless link properties (e.g., reliability) according to application data traffic patterns. TLR constructs a reliable and efficient messaging structure for data packet flow, based on which other high-level logics for further enhancing messaging reliability and efficiency can be applied. TLR is generic to all sensornet applications and can be performed automatically without any explicit input from applications.
  - At the higher level, we identify the components *application-adaptive structuring* (AST) and *application-adaptive scheduling* (ASC) that control the spatial and temporal flow of data packets to facilitate functionalities (e.g., in-network processing and transport control) that are tightly coupled with applications. AST and ASC incorporate application-specific properties (e.g., methods of in-network processing and application QoS requirements) in forming messaging structures and in scheduling packet transmissions respectively, to improve the reliability and efficiency of sensornet messaging.

SMA provides the guidelines for composing sensornet messaging services and for designing the individual messaging components.

- **[Link estimation & routing.]** To build the basic structure for reliable and efficient messaging, and to provide an instantiation of the TLR component of SMA, we propose the routing protocol *Learn on the Fly* (LOF) [102]. LOF estimates link quality based on data transmissions, and it chooses routes by way of a locally measurable metric ELD, the *expected MAC latency per unit-distance to the destination*. In addressing the challenge of data-driven link estimation to routing protocol design, i.e., uneven link sampling where a used link is continuously sampled yet an unused link never gets sampled, LOF uses the technique of exploratory neighbor sampling where every alternative link (or route) is sampled with controlled probability and frequency.

  Using an event traffic trace from the field sensornet of ExScal [12], we experimentally evaluate the design and the performance of LOF in a testbed of 195 Stargates [1] with 802.11b radios (which are usually used in sensornet backbones). We also compare the performance of LOF with that of existing protocols, represented by the geography-unaware ETX [27, 95] and the geography-based PRD [83]. We find that LOF reduces end-to-end MAC latency, reduces energy consumption in packet delivery, and improves route stability. Besides bursty event traffic, we evaluate LOF in the case of periodic traffic, and we find that LOF outperforms existing protocols in that case too. The results corroborate the feasibility as well as the benefits of data-driven link estimation and routing.

- **[Packing-oriented scheduling.]** To understand the algorithmic issues in application-adaptive scheduling and to provide an instantiation of the ASC component of SMA, we study the ASC component in detail by taking packet packing (i.e., aggregating shorter packets into longer ones) as an example of in-networking processing [103]. We propose to schedule packet transmissions so that the *utility* of a transmission (e.g., degree of in-network aggregation) is maximized. To this end. we propose a distributed algorithm in which a node dynamically estimates the potential utility of transmitting a packet and decides when to transmit so that the utility is maximized while satisfying certain end-to-end timeliness guarantees on data delivery. This algorithmic framework is generically applicable to other in-networking processing methods.

  We evaluate our design via both simulation and experimentation with Tmote Sky sensor nodes. We find that our approach significantly improves energy efficiency and messaging reliability. For instance, the energy efficiency is improved by a factor up to 3.22, and the reliability is improved by 12.92%.

- **[Reliable & real-time data transport.]** Besides facilitating in-network processing, another important role of the ASC component is to support the application QoS requirements such as reliability and timeliness in data delivery. In the context of event-detection applications, we propose the protocol *Reliable Bursty Convergecast* (RBC) for reliable and real-time data transport in sensornets [100].

  We study the limitations of two commonly used hop-by-hop packet recovery schemes in bursty convergecast. We discover that the lack of retransmission scheduling in both schemes makes retransmission-based packet recovery ineffective in the case of bursty convergecast. Moreover, in-order packet delivery makes the communication channel under-utilized in the presence of packet- and ack-loss.

To address the challenges, we design protocol RBC (for Reliable Bursty Convergecast). Taking advantage of the unique sensor network models, RBC features the following mechanisms:

- To improve channel utilization, RBC uses a window-less block acknowledgment scheme that enables continuous packet forwarding in the presence of packet- and ack-loss. The block acknowledgment also reduces the probability of ack-loss, by replicating the acknowledgment for a received packet.
- To ameliorate retransmission-incurred channel contention, RBC introduces differentiated contention control, which ranks nodes by their queuing conditions as well as the number of times that the enqueued packets have been transmitted. A node ranked the highest within its neighborhood accesses the channel first.

In addition, we design techniques that address the challenges of timer-based retransmission control in bursty convergecast:

- To deal with continuously changing ack-delay, RBC uses adaptive retransmission timer which adjusts itself as network state changes.
- To reduce delay in timer-based retransmission and to expedite retransmission of lost packets, RBC uses block-NACK, retransmission timer reset, and channel utilization protection.

We evaluate RBC by experimenting with an outdoor testbed of 49 MICA2 motes and with realistic traffic trace from the field sensor network of Lites. Our experimental results show that, compared with a commonly used implicit-ack scheme, RBC increases the packet delivery ratio by a factor of 2.05 and reduces the packet delivery delay by a factor of 10.91. Moreover, RBC achieves a goodput of 6.37 packets/second for the traffic trace of Lites, almost reaching the optimal goodput — 6.66 packets/second — for the trace.

In addition to addressing the challenges of wireless communication, resource constraints, and application diversity, we also address the following foundational and algorithmic issues for scalable dependability in sensornet messaging:

- **[Local stabilization.]** For scalable dependability in large scale dynamic sensornets, it is desirable that faults be contained locally around where they have occurred, and that the time taken for a system to stabilize be a function $\mathcal{F}$ of the size of the fault-perturbed regions instead of the size of the system. We call this property *$\mathcal{F}$-local stabilization*. To characterize the properties of locally stabilizing systems, we formulate the concepts of perturbation size, $\mathcal{F}$-local stabilization, and range of contamination [15]. These concepts take into account the minimum amount of work required for systems to stabilize and are generically applicable to networking as well as distributed computing problems.

- **[Locally-stabilizing routing.]** For shortest path routing, a basic problem in messaging, we design a locally-stabilizing protocol LSRP (for Locally Stabilizing shortest path Routing Protocol) [15]. LSRP achieves local stabilization via two techniques. Firstly, it layers system computation into three diffusing waves each having a different propagation speed, i.e., "stabilization wave" with the lowest speed, "containment wave" with intermediate speed, and "super-containment wave" with the highest speed. The containment wave contains the mistakenly initiated stabilization wave, the super-containment wave contains the mistakenly initiated containment wave, and the super-containment wave self-stabilizes itself locally. Secondly, LSRP avoids forming loops during stabilization, and it removes all transient loops within small constant time.

Upon starting at an arbitrary state where the perturbation size is $p$, LSRP stabilizes to yield shortest path routes within $O(p)$ time, and the nodes affected by the perturbation are within $O(p)$ distance from the perturbed regions. Given two (or more) perturbed regions, LSRP stabilizes each region independently of and concurrently with the other(s) if the half distance between the regions is $\omega(p')$, where $p'$ is the size of the largest perturbed region. Moreover, LSRP not only guarantees loop-freedom during stabilization, it also removes any existing loop (which is created by a fault) within constant time irrespective of the loop length. To the best of our knowledge, LSRP is the first protocol that achieves local stabilization in shortest path routing.

Besides analysis and simulation, most algorithms proposed in the dissertation have been implemented and have provided dependable messaging services for real-world sensornet systems such as *A Line in the Sand* [14], *ExScal* [12], and *MUSE* [80].

## 1.3 Organization of the dissertation

The rest of this dissertation is organized as follows. We present the sensornet messaging architecture SMA in Chapter 2, followed by the discussion of the individual components of SMA from Chapter 3 to Chapter 5. More specifically, we discuss data-driven link estimation and routing in sensornets in Chapter 3, we discuss application-adaptive scheduling in Chapter 4, and we present the protocol for reliable and real-time data transport in Chapter 5.

For scalable dependability in messaging, we present the concept and protocol for locally-stabilizing shortest path routing in Chapter 6. We discuss related work in Chapter 7, and we make concluding remarks in Chapter 8.

# CHAPTER 2

# SENSORNET MESSAGING ARCHITECTURE

To deal with the complex dynamics of wireless communication and to support diversified applications in a scalable manner, it is desirable to have a unified messaging architecture that identifies the common components as well as their interactions [103]. To this end, we first review the basic functions of sensornet messaging, based upon which we identify the common messaging components and design the messaging architecture SMA.

## 2.1 Basic components of sensornet messaging

As in the case for the Internet, the objective of messaging in sensornets is to deliver data from their sources to their destinations. To this end, the basic tasks of messaging are, given certain QoS constraints (e.g., reliability and latency) on data delivery, choose the route(s) from every source to the corresponding destination(s) and schedule packet flow along the route(s). As we argued before, unlike wired networks, the design of messaging in sensornets is challenging as a result of wireless communication dynamics, resource constraints, and application diversity.

Given the complex dynamics of sensornet wireless links, a key component of sensornet messaging is precisely estimating wireless link properties and then finding routes of high quality links to deliver data traffic. Given that data traffic pattern affects wireless link properties due to interference among simultaneous transmissions [102], link estimation and routing should be able to take into account the impact of application data traffic, and we call this basic messaging component *traffic-adaptive link estimation and routing (TLR)*.

With the basic communication structure provided by the TLR component, another important task of messaging is to adapt the structure and data transmission schedules according to application properties such as in-network processing and QoS requirements. Given the resource constraints in sensornets, application data may be processed in the network before it reaches the final destination to improve resource utilization (e.g., to save energy and to reduce data traffic load). For instance, data arriving from different sources may be compressed at an intermediate node before it is forwarded further. Given that messaging determines the spatial and temporal flow of application data and that data items from different sources can be processed together only if they meet somewhere in the network, messaging significantly affects the degree of processing achievable in the network [103, 34]. It is therefore desirable that messaging consider in-network processing when deciding how to form the messaging structure and how to schedule data transmissions. In addition, messaging should also consider application QoS requirements (e.g., reliability and latency in packet delivery), because messaging structure and transmission schedule determine the QoS experienced by application traffic [54, 90, 47]. In-network processing and QoS requirements tend to be tightly coupled with applications, thus we call the structuring and scheduling in messaging *application-adaptive structuring (AST)* and *application-adaptive scheduling (ASC)* respectively.

## 2.2 SMA: an architecture for sensornet messaging

The messaging components discussed in the previous section are coupled with wireless communication and applications in different ways and at different degrees, thus we adopt two levels of abstraction in designing the architecture for sensornet messaging. The architecture, SMA (for *Sensornet Messaging Architecture*), is shown in Figure 2.1. At the lower

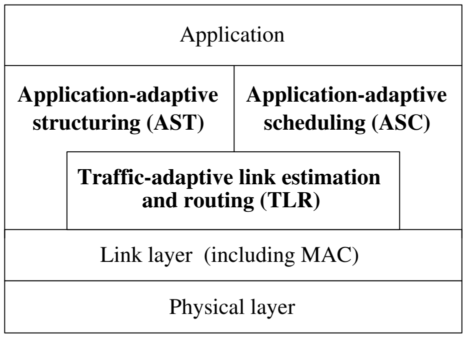

Application
Application-adaptive structuring (AST)
Application-adaptive scheduling (ASC)
Traffic-adaptive link estimation and routing (TLR)
Link layer (including MAC)
Physical layer

Figure 2.1: SMA: a sensornet messaging architecture

level, traffic-adaptive link estimation and routing (TLR) interacts directly with the link layer to estimate link properties and to form the basic routing structure in a traffic-adaptive manner. TLR can be performed without explicit input from applications, and TLR does not directly interface with applications. At the higher level, both application-adaptive structuring (AST) and application-adaptive scheduling (ASC) need input from applications, thus AST and ASC interface directly with applications. Besides interacting with TLR, AST and ASC may need to directly interact with link layer to perform tasks such as adjusting radio transmission power level and fetching link-layer acknowledgment to a packet transmission.In the architecture, the link and physical layers support higher-layer messaging tasks (i.e., TLR, AST, and ASC) by providing the capability of communication within one-hop neighborhoods.

In what follows, we elaborate on the individual components of SMA.

### 2.2.1 TLR: traffic-adaptive link estimation and routing

To estimate wireless link properties, one approach is to use beacon packets as the basis of link estimation. That is, neighbors exchange broadcast beacons, and they estimate broadcast link properties based on the quality of receiving one another's beacons (e.g., the ratio of beacons successfully received, or the RSSI/LQI of packet reception); then, neighbors estimate unicast link properties based on those of beacon broadcast, since data are usually transmitted via unicast. This approach of beacon-based link estimation has been used in several routing protocols including ETX [95, 27].

We find that there are two major drawbacks of beacon-based link estimation. Firstly, it is hard to build high-fidelity models for temporal correlations in link properties [93, 91, 55], thus most existing routing protocols do not consider temporal link properties and assume instead independent bit error or packet loss. Consequently, significant estimation error can be incurred, as we show in [102]. Secondly, even if we could precisely estimate unicast link properties, the estimated values may only reflect unicast properties in the absence — instead of the presence— of data traffic, which matters since the network traffic affects link properties due to interference. This is especially the case in event-detection applications, where events are usually rare (e.g., one event per day) and tend to last only for a short time at each network location (e.g., less than 20 seconds). Therefore, beacon-based link estimation cannot precisely estimate link properties in a traffic-adaptive manner.

To address the limitations of beacon-based link estimation, Zhang et al [102] propose the LOF routing protocol (for *Learn on the Fly*) that estimates unicast link properties via MAC feedback[1] for data transmissions themselves without using beacons. Since MAC feedback reflects in-situ the network condition in the presence of application traffic, link estimation in LOF is traffic-adaptive. LOF also addresses the challenges of data-driven link estimation to routing protocol design, such as uneven link sampling (i.e., the quality of a link is not sampled unless the link is used in data forwarding). It has been shown that, compared with beacon-based link estimation and routing, LOF improves both the reliability and energy efficiency in data delivery. More importantly, LOF quickly adapts to changing traffic patterns, and this is achieved without any explicit input from applications.

The TLR component provides the basic service of automatically adapting link estimation and routing structure to application traffic patterns. TLR also exposes its knowledge of link and route properties (such as end-to-end packet delivery latency) to higher level components AST and ASC, so that AST and ASC can optimize the degree of in-network processing while providing the required QoS in delivering individual pieces of application data.

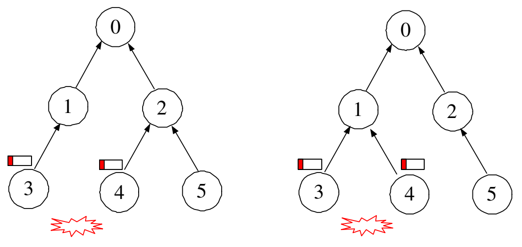
### 2.2.2 AST: application-adaptive structuring

One example of application-adaptive structuring is to adjust messaging structure according to application QoS requirements. For instance, radio transmission power level determines the communication range of each node and the connectivity of a network. Accordingly, transmission power level affects the number of routing hops between any pairs of source and destination and thus packet delivery latency. Transmission power level also

[1]The MAC feedback for a unicast transmission includes whether the transmission has succeeded and how many times the packet has been retransmitted at the MAC layer.

determines the interference range of packet transmissions, and thus it affects packet delivery reliability. Therefore, radio transmission power level (and thus messaging structure) can be adapted to satisfy specific application QoS requirements, and Kawadia and Kumar have studied this in [54].

Besides QoS-oriented structuring, another example of application-adaptive structuring is to adjust messaging structure according to the opportunities of in-network processing. Messaging structure determines how data flows spatially, and thus affects the degree of in-network processing achievable. For instance, as shown in Figure 2.2(a), nodes 3 and 4



(a) Before adaptation (b) After adaptation

Figure 2.2: Example of application-adaptive structuring

detect the same event simultaneously. But the detection packets generated by nodes 3 and 4 cannot be aggregated in the network, since they follow different routes to the destination node 0. On the other hand, if node 4 can detect the correlation between its own packet and that generated by node 3, node 4 can change its next-hop forwarder to node 1, as shown in Figure 2.2(b). Then the packets generated by nodes 3 and 4 can meet at node 1, and be aggregated before being forwarded to the destination node 0.

In general, to improve the degree of in-network processing, a node should consider the potential in-network processing achievable when choosing the next-hop forwarder. One way to realize this objective is to adapt the existing routing metric. For each neighbor $k$, a node $j$ estimates the utility $u_{j,k}$ of forwarding packets to $k$, where the utility is defined as the reduction in messaging cost (e.g., number of transmissions) if $j$'s packets are aggregated with $k$'s packets. Then, if the cost of messaging via $k$ without aggregation is $c_{j,k}$, the associated messaging cost $c'_{j,k}$ can be adjusted as follows (to reflect the utility of in-network processing):

$$c'_{j,k} = c_{j,k} - u_{j,k}$$

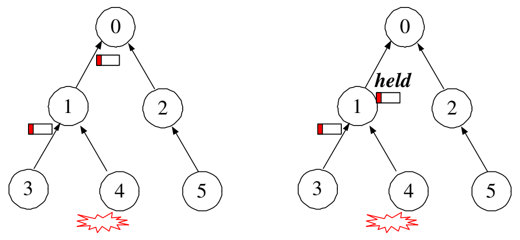
Accordingly, a neighbor with the lowest adjusted messaging cost is selected as the next-hop forwarder.

Since QoS requirements and in-network processing vary from one application to another, AST needs input from applications, and it needs to interface with applications directly.

### 2.2.3 ASC: application-adaptive scheduling

One example of application-adaptive scheduling is to schedule packet transmissions to satisfy certain application QoS requirements. To improve packet delivery reliability, for instance, lost packets can be retransmitted. But packet retransmission consumes energy, and not every sensornet application needs 100% packet delivery rate. Therefore, the number of retransmissions can be adapted to provide different end-to-end packet delivery rates while minimizing the total number of packet transmissions [14]. To provide differentiated timeliness guarantee on packet delivery latency, we can also introduce priority in transmission scheduling such that urgent packets have high priority of being transmitted [100]. Similarly, data streams from different applications can be ranked so that transmission scheduling ensures differentiated end-to-end throughput to different applications [32].

Besides QoS-oriented scheduling, another example of application-adaptive scheduling to schedule packet transmissions according to the opportunities of in-network processing. Given a messaging structure formation, transmission scheduling determines how data flows along the structure temporally and thus the degree of in-network processing achievable. To give an example, let us look at Figure 2.3(a). A dupers node 4 detects an event earlier than



(a) Before adaptation

(b) After adaptation

Figure 2.3: Example of application-adaptive scheduling

node 3 does. Then the detection packet from node 4 can reach node 1 earlier than the packet from node 3. If node 1 immediately forwards the packet from node 4 after receiving it, then the packet from node 4 cannot be aggregated with that from node 3, since the packet from node 4 has already left node 1 when the packet from node 3 reaches node 1. On the other hand, if node 1 is aware of the correlation between packets from nodes 3 and 4, then node 1 can hold the packet from 4 after receiving it (as shown in Figure 2.3(b)). Accordingly, the packet from node 3 can meet that from node 4, and these packets can be aggregated before being forwarded.

In general, a node should consider both application QoS requirements and the potential in-network processing when scheduling data transmissions, so that application QoS requirements are better satisfied and the degree of in-network processing is improved. Given that in-network processing and QoS requirements are application specific, ASC needs to directly interface with applications.

**Remark.** It is desirable that the components TLR, AST, and ASC be deployed all together to achieve the maximal network performance. That said, the three components can also be deployed in an incremental manner while maintaining the benefits of each individual component, as shown in [102] and [103].

## 2.3 Summary

To address the challenges of wireless communication, constrained resources, and application diversity in sensornets, we propose the *Sensornet Messaging Architecture* (SMA) in which we adopt two levels of abstraction:

- At the lower level, we identify the component *traffic-adaptive link estimation and routing* (TLR) that is responsible for precisely estimating wireless link properties (e.g., reliability) according to application data traffic patterns. TLR is generic to all sensornet applications and can be performed automatically without explicit input from applications.

- At the higher level, we identify the components *application-adaptive structuring* (AST) and *application-adaptive scheduling* (ASC) to support functionalities (e.g., in-network processing and QoS) that are tightly coupled with applications. AST and

ASC incorporate application-specific properties (e.g., methods of in-network processing and QoS requirements) in forming messaging structures and in scheduling packet transmissions respectively.

In the immediately following chapters of this dissertation, we first discuss the routing protocol LOF, an instantiation of the component TLR, in Chapter 3. Then, we discuss the ASC component from the perspective of in-network processing in Chapter 4, and from the perspective of application QoS requirements in Chapter 5. Detailed study of AST is a part of our future work.

# CHAPTER 3

# DATA-DRIVEN LINK ESTIMATION AND ROUTING

In this chapter, we study in detail why and how to perform link estimation and routing in a traffic-adaptive manner, and we present the routing protocol *Learn on the Fly* (LOF) as an instantiation of the TLR component of SMA.

## 3.1 Motivation

As the quality of wireless links, for instance, packet delivery rate, varies both temporally and spatially in a complex manner [10, 56, 107], estimating link quality is an important aspect of routing in wireless networks. To this end, peers exchange broadcast beacons periodically in existing routing protocols [27, 30, 31, 83, 95], and the measured quality of broadcast acts as the basis of link estimation. Nonetheless, beacon-based link estimation has several drawbacks:

- Firstly, link quality for broadcast beacons differs significantly from that for unicast data, because broadcast beacons and unicast data differ in packet size, transmission rate, and coordination method at the media-access-control (MAC) layer [22, 66]. Therefore, we have to estimate unicast link quality based on that of broadcast.
- It is, however, difficult to precisely estimate unicast link quality via that of broadcast, because temporal correlations of link quality assume complex patterns [94] and are

hard to model. As a result, existing routing protocols do not consider temporal link properties in beacon-based estimation [27, 95]. Thus the link quality estimated using periodic beacon exchange may not accurately apply for unicast data, which can negatively impact the performance of routing protocols.

- Even if we could precisely estimate unicast link quality based on that of broadcast, beacon-based link estimation may not reflect in-situ network condition either. For instance, a typical application of sensornets is to monitor an environment (be it an agricultural field or a classified area) for events of interest to the users. Usually, the events are rare. Yet when an event occurs, a large burst of data packets is often generated that needs to be routed reliably and in real-time to a base station [100]. In this context, even if there were no discrepancy between the actual and the estimated link quality using periodic beacon exchange, the estimates still tend to reflect link quality in the absence, rather than in the presence, of bursty data traffic. This is because: Firstly, link quality changes significantly when traffic pattern changes (as we will show in Section 3.2.2); Secondly, link quality estimation takes time to converge, yet different bursts of data traffic are well separated in time, and each burst lasts only for a short period.

Beacon-based link estimation is not only limited in reflecting the actual network condition, it is also inefficient in energy usage. In existing routing protocols that use link quality estimation, beacons are exchanged periodically. Therefore, energy is consumed unnecessarily for the periodic beaconing when there is no data traffic. This is especially true if the events of interest are infrequent enough that there is no data traffic in the network most of the time [100].

To deal with the shortcomings of beacon-based link quality estimation and to avoid unnecessary beaconing, we propose the routing protocol LOF that uses data transmission itself as the basis of link estimation and thus is traffic-adaptive.

In the remainder of this chapter, we study in detail the shortcomings of beacon-based link quality estimation, and we analyze the feasibility of data-driven routing in Section 3.2. Following that, we present the routing metric ELD in Section 3.3, and we design the protocol LOF in Section 3.4. We experimentally evaluate LOF in Section 3.5, and we summarize this chapter in Section 3.6.

## 3.2 Why data-driven link estimation and routing?

In this section, we first experimentally study the impact of packet type, packet length, and interference on link properties[2]. Then we discuss the shortcomings of beacon-based link property estimation, as well as the concept of data-driven link estimation and routing.

### 3.2.1 Experiment design

We set up two 802.11b network testbeds as follows.

**Outdoor testbed.** In an open field (see Figure 3.1), we deploy 29 Stargates in a straight line, with a 45-meter separation between any two consecutive Stargates. The Stargates run Linux with kernel 2.4.19. Each Stargate is equipped with a SMC 2.4GHz 802.11b wireless card and a 9dBi high-gain collinear omnidirectional antenna, which is raised 1.5 meters above the ground. To control the maximum communication range, the transmission power level of each Stargate is set as 35. (Transmission power level is a tunable parameter for

[2]In this chapter, the phrases *link quality* and *link property* are used interchangeably.

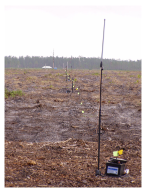
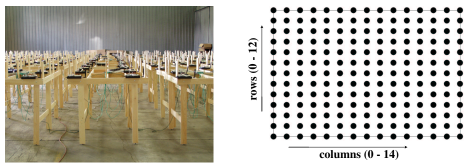


Figure 3.1: Outdoor testbed

802.11b wireless cards, and its range is 127, 126, . . . , 0, 255, 254, . . . , 129, 128, with 127 being the lowest and 128 being the highest.)

**Sensornet testbed Kansei.** In an open warehouse with flat aluminum walls (see Figure 3.2(a)), we deploy 195 Stargates in a 15 $\times$ 13 grid (as shown in Figure 3.2(b)) where the separation between neighboring grid points is 0.91 meter (i.e., 3 feet). The deployment



(a) Kansei (b) grid topology

Figure 3.2: Sensornet testbed Kansei

is a part of the sensornet testbed Kansei [13]. For convenience, we number the rows of the grid as 0 - 12 from the bottom up, and the columns as 0 - 14 from the left to the right. Each Stargate is equipped with the same SMC wireless card as in the outdoor testbed. To create realistic multi-hop wireless networks similar to the outdoor testbed, each Stargate is equipped a 2.2dBi rubber duck omnidirectional antenna and a 20dB attenuator. We raise the Stargates 1.01 meters above the ground by putting them on wood racks. The transmission power level of each Stargate is set as 60, to simulate the low-to-medium density multi-hop networks where a node can reliably communicate with around 15 neighbors.

The Stargates in the indoor testbed are equipped with wall-power and outband Ethernet connections, which facilitate long-duration complex experiments at low cost. We use the indoor testbed for most of the experiments in this chapter; we use the outdoor testbed mainly for justifying the generality of the phenomena observed in the indoor testbed.

**Experiments.** In the *outdoor testbed*, the Stargate at one end acts as the sender, and the other Stargates act as receivers. Given the constraints of time and experiment control, we leave complex experiments to the indoor testbed and only perform relatively simple experiments in the outdoor testbed: the sender first sends 30,000 1200-byte broadcast packets, then it sends 30,000 1200-byte unicast packets to each of the receivers.

In the *indoor testbed*, we let the Stargate at column 0 of row 6 be the sender, and the other Stargates in row 6 act as receivers. To study the impact of interference, we consider the following scenarios (which are named according to the interference):

- *Interferer-free*: there is no interfering transmission. The sender first sends 30,000 broadcast packets each of 1200 bytes, then it sends 30,000 1200-byte unicast packets to each of the receivers, and lastly it broadcasts 30,000 30-byte packets.

- *Interferer-close*: one "interfering" Stargate at column 0 of row 5 keeps sending 1200-byte unicast packets to the Stargate at column 0 of row 7, serving as the source of the interfering traffic. The sender first sends 30,000 1200-byte broadcast packets, then it sends 30,000 1200-byte unicast packets to each of the receivers.

- *Interferer-middle*: the Stargate at column 7 of row 5 keeps sending 1200-byte unicast packets to the Stargate at column 7 of row 7. The sender performs the same as in the case of interferer-close.

- *Interferer-far*: the Stargate at column 14 of row 5 keeps sending 1200-byte unicast packets to the Stargate at column 14 of row 7. The sender performs the same as in the case of interferer-close.

- *Interferer-exscal*: In generating the interfering traffic, every Stargate runs the routing protocol LOF (as detailed in later sections of this chapter), and the Stargate at the upper-right corner keeps sending packets to the Stargate at the left-bottom corner, according to an event traffic trace from the field sensornet of ExScal [12] . The traffic trace corresponds to the packets generated by a Stargate when a vehicle passes across the corresponding section of ExScal network. In the trace, 19 packets are generated, with the first 9 packets corresponding to the start of the event detection and the last 10 packets corresponding to the end of the event detection. Figure 3.3 shows, in sequence, the intervals between packets 1 and 2, 2 and 3, and so on. The

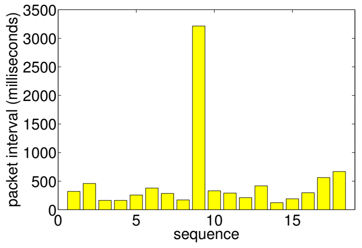


Figure 3.3: The traffic trace of an ExScal event

sender performs the same as in the case of interferer-close.

In all of these experiments, except for the case of interferer-exscal, the packet generation frequency, for both the sender and the interferer, is 1 packet every 20 milliseconds. In the case of interferer-exscal, the sender still generates 1 packet every 20 milliseconds, yet the interferer generates packets according to the event traffic trace from ExScal, with the inter-event-run interval being 10 seconds. (Note that the scenarios above are far from being complete, but they do give us a sense of how different interfering patterns affect link properties.)

In the experiments, broadcast packets are transmitted at the basic rate of 1M bps, as specified by the 802.11b standard. Not focusing on the impact of packet rate in our study, we set unicast transmission rate to a fixed value (e.g., 5.5M bps). (We have tested different unicast transmission rates and observed similar phenomena.) For other 802.11b parameters, we use the default configuration that comes with the system software. For instance, unicast transmissions use RTS-CTS handshake, and each unicast packet is retransmitted up to 7 times until success or failure in the end.

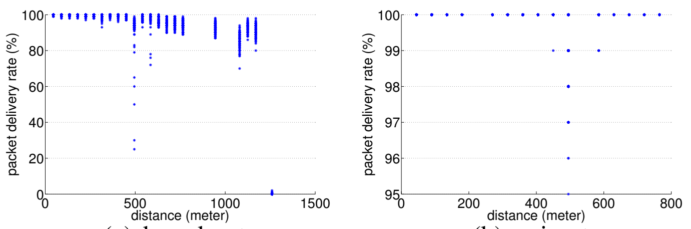
### 3.2.2 Experimental results

For each case, we measure various link properties, such as packet delivery rate and the run length of packets successfully received without any loss in between, for each link defined by the sender - receiver. Due to space limitations, however, we only present the data on packet delivery rate here. The packet delivery rate is calculated once every 100 packets (we have also calculated delivery rates in other granularities, such as once every 20, 50 or 1000 packets, and similar phenomena were observed).

We first present the difference between broadcast and unicast when there is no interference, then we present the impact of interference on network conditions as well as the difference between broadcast and unicast.

## Interferer free

Figure 3.4 shows the scatter plot of the delivery rates for broadcast and unicast packets



(a) broadcast (b) unicast

Figure 3.4: Outdoor testbed

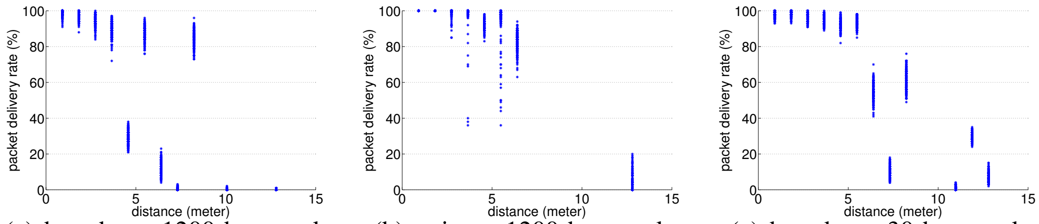
at different distances in the outdoor testbed. From the figure, we observe the following:

- Broadcast has longer communication range than unicast. This is due to the fact that the transmission rate for broadcast is lower, and that there is no RTS-CTS handshake for broadcast. (Note: the failure in RTS-CTS handshake also causes a unicast to fail.)

- For links where unicast has non-zero delivery rate, the mean delivery rate of unicast is higher than that of broadcast. This is due to the fact that each unicast packet is retransmitted up to 7 times upon failure.

- The variance in packet delivery rate is lower in unicast than that in broadcast. This is due to the fact that unicast packets are retransmitted upon failure, and the fact that there is RTS-CTS handshake for unicast. (Note: the success in RTS-CTS handshake implies higher probability of a successful unicast, due to temporal correlations in link properties [21].)

Similar results are observed in the indoor testbed, as shown in Figures 3.5(a) and 3.5(b). Nevertheless, there are exceptions at distances 3.64 meters and 5.46 meters, where the



(a) broadcast: 1200-byte packet (b) unicast: 1200-byte packet (c) broadcast: 30-byte packet

Figure 3.5: Indoor testbed Kansei

delivery rate of unicast takes a wider range than that of broadcast. This is likely due to temporal changes in the environment. Comparing Figures 3.5(a) and 3.5(c), we see that packet length also has significant impact on the mean and variance of packet delivery rate.

**Implication.** From Figures 3.4 and 3.5, we see that packet delivery rate differs significantly between broadcast and unicast, and the difference varies with environment, hardware, and packet length.

### Interference scenarios

To demonstrate how network condition changes with interference scenarios, Figure 3.6 shows the broadcast packet delivery rates in different interference scenarios. We see that

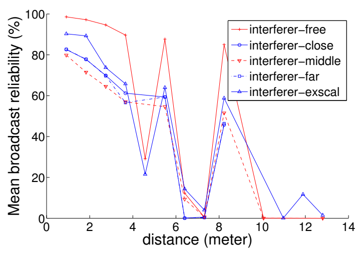


Figure 3.6: Network condition, measured in broadcast reliability, in different interference scenarios

broadcast packet delivery rate varies significantly (e.g., up to 39.26%) as interference patterns change. Thus, link properties estimated for one scenario may not apply to another.

Having shown the impact of interference patterns on network condition, Figure 3.7 shows how the difference between broadcast and unicast in the mean packet delivery rate

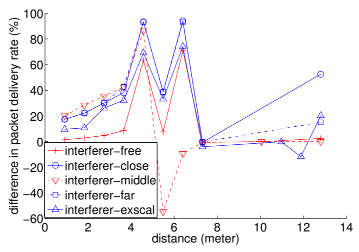


Figure 3.7: The difference between broadcast and unicast in different interference scenarios

changes as the interference and distance change. Given a distance and an interference scenario, the difference is calculated as $\frac{U-B}{B}$, where $U$ and $B$ denote the mean delivery rate for unicast and broadcast respectively. From the figure, we see that the difference is significant (up to 94.06%), and that the difference varies with distance. Moreover, the difference changes significantly (up to 103.41%) as interference pattern changes.

**Implication.** For sensornets where data bursts are well separated in time and possibly in space (e.g., in bursty convergecast), the link properties experienced by periodic beacons may well differ from those experienced by data traffic. Moreover, the difference between broadcast and unicast changes as interference pattern changes.

### 3.2.3 Data-driven routing

To ameliorate the differences between broadcast and unicast link properties, researchers have proposed to make the length and transmission rate of broadcast beacons be the same as those of data packets, and then estimate link properties of unicast data via those of broadcast beacons by taking into account factors such as link asymmetry. ETX [27] has explored this approach. Nevertheless, this approach may not be always feasible when the length of data packets is changing; or even if the approach is always feasible, it still does not guarantee that link properties experienced by periodic beacons reflect those in the presence of data traffic, especially in event-driven sensornet applications. Moreover, the existing method for estimating metrics such as ETX does not take into account the temporal correlations in link properties [21] (partly due to the difficulty of modeling the temporal correlations themselves [94]), which further decreases its estimation fidelity. For instance, Figure 3.8 shows the significant error[3] in estimating unicast delivery rate via that of broadcast under different interference scenarios when temporal correlations in link properties are not considered (i.e., assuming independent bit error and packet loss). Therefore, it is not trivial, if

[3]The error is defined as actual unicast link reliability minus the estimated link reliability

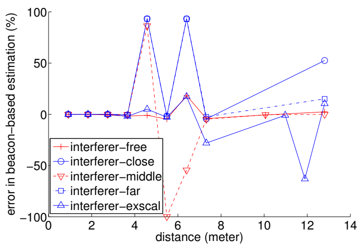


Figure 3.8: Error in estimating unicast delivery rate via that of broadcast

even possible, to precisely estimate link properties for unicast data via those of broadcast beacons.

To circumvent the difficulty of estimating unicast link properties via those of broadcast, we propose to directly estimate unicast link properties via data traffic itself. In this context, since we are not using beacons for link property estimation, we also explore the idea of not using periodic beacons in routing at all (i.e., beacon-free routing) to save energy; otherwise, beaconing requires nodes to wake up periodically even when there is no data traffic.

To enable data-driven routing, we need to find alternative mechanisms for accomplishing the tasks that are traditionally assumed by beacons: acting as the basis for link property estimation, and diffusing information (e.g., the cumulative ETX metric). In sensornet backbones, data-driven routing is feasible because of the following facts:

- **MAC feedback.** In MACs where every frame transmission is acknowledged by the receiver (e.g., in the 802.11b MAC), the sender can determine if a transmission has succeeded by checking whether it receives the acknowledgment. Also, the sender can determine how long each transmission takes, i.e., MAC latency. Therefore, the sender is able to get information on link properties without using any beacons. (Note: it has also been shown that MAC latency is a good routing metric for optimizing wireless network throughput [17].)

- **Mostly static network & geography.** Nodes are static most of the time, and their geographic locations are readily available via devices such as GPS. Therefore, we can use geography-based routing in which a node only needs to know the location of the destination and the information regarding its local neighborhood (such as the quality of the links to its neighbors). Thus, only the location of the destination (e.g., the base station in convergecast) needs to be diffused across the network. Unlike in beacon-based distance-vector routing, the diffusion happens infrequently since the destination is static most of the time. In general, control packets are needed only when the location of a node changes, which occurs infrequently.

In what follows, we first present the routing metric ELD which is based on geography and MAC latency, then we present the design of LOF which implements ELD without using periodic beacons.

**Remarks**:

- Although parameters such as Receiver Signal Strength Indicator (RSSI), Link Quality Indicator (LQI), and Signal to Noise Ratio (SNR) also reflect link reliability, it is difficult to use them as a precise prediction tool [10]. Moreover, the aforementioned parameters can be fetched only at packet receivers (instead of senders), and extra control packets are needed to convey these information back to the senders if we want to use them as the basis of link property estimation. Therefore, we do not recommend using these parameters as the core basis of data-driven routing, especially when senders need to precisely estimate in-situ link properties.

- Our objective in this chapter is to explore the idea of data-driven link property estimation and routing, and it is not our objective to prove that geography-based routing is better than distance-vector routing. In principle, we could have used distance-vector routing together with data-driven link property estimation, but this would introduce periodic control packets which we would like to avoid to save energy. We will show in Section 3.5.3, however, that geography-based data-driven routing has similar performance as that of distance-vector data-driven routing.

- Conceptually, we could have also defined our routing metric based on other parameters such as ETX [27] or RNP [21]. Nevertheless, the firmware of our SMC WLAN cards does not expose information on the number of retries of a unicast transmission, which makes it hard to estimate ETX or RNP directly via data traffic. As a part of our future work, we plan to design mechanisms to estimate ETX and RNP via data traffic (e.g., in IEEE 802.15.4 based mote networks) and study the corresponding protocol performance.

## 3.3 ELD: the routing metric

In this section, we first formulate the routing metric ELD, the *expected MAC latency per unit-distance to the destination*, then we analyze the sample size requirement in routing.

### 3.3.1 A metric using MAC latency and geography

For convergecast in sensornets (especially for event-driven applications), packets need to be routed reliably and in real-time to the base station. As usual, packets should also be delivered in an energy-efficient manner. Therefore, a routing metric should reflect link reliability, packet delivery latency, and energy consumption at the same time. One such

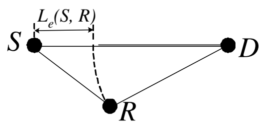


Figure 3.9: $L_e$ calculation

metric that we adopt in LOF is based on MAC latency, i.e., the time taken for the MAC to transmit a data frame. (We have mathematically analyzed the relationship among MAC latency, energy consumption, and link reliability, and we find that MAC latency is strongly related to energy consumption in a positive manner, and the ratio between them changes only slightly as link reliability changes. Thus, routing metrics optimizing MAC latency would also optimize energy efficiency. Interested readers can find the detailed analysis in [101].)

Given that MAC latency is a good basis for route selection and that geography enables low frequency information diffusion, we define a routing metric ELD, *the expected MAC latency per unit-distance to the destination*, which is based on both MAC latency and geography. Specifically, given a sender $S$, a neighbor $R$ of $S$, and the destination $D$ as shown in Figure 3.9, we first calculate the *effective geographic progress* from $S$ to $D$ via $R$, denoted by $L_e(S, R)$, as $(L_{S,D} - L_{R,D})$, where $L_{S,D}$ denotes the distance between S and D, and $L_{R,D}$ denotes the distance between R and D. Then, we calculate, for the sender $S$, the *MAC latency per unit-distance to the destination* (LD) via $R$, denoted by $LD(S, R)$, as[4]

$$\begin{cases} \frac{D_{S,R}}{L_e(S,R)} & \text{if } L_{S,D} > L_{R,D} \\ \infty & \text{otherwise} \end{cases} \tag{3.1}$$

[4]Currently, we focus on the case where a node forwards packets only to a neighbor closer to the destination than itself.

where $D_{S,R}$ is the MAC latency from $S$ to $R$. Therefore, the ELD via $R$, denoted as $ELD(S, R)$, is $E(LD(S, R))$ which is calculated as

$$\begin{cases} \frac{E(D_{S,R})}{L_e(S,R)} & \text{if } L_{S,D} > L_{R,D} \\ \infty & \text{otherwise} \end{cases} \tag{3.2}$$

For every neighbor $R$ of $S$, $S$ associates with $R$ a rank

$$\langle ELD(S, R), var(LD(S, R)), L_{R,D}, ID(R)\rangle$$

where $var(LD(S, R))$ denotes the variance of $LD(S, R)$, and $ID(R)$ denotes the unique ID of node $R$. Then, $S$ selects as its next-hop forwarder the neighbor that ranks the lowest among all the neighbors. (Note: routing via metric ELD is a greedy approach, where each node tries to optimize the local objective. Like many other greedy algorithms, this method is effective in practice, as shown via experiments in Section 3.5.)

To understand what ELD implies in practice, we set up an experiment as follows: consider a line network formed by row 6 of the indoor testbed shown in Figure 3.2, the Stargate $S$ at column 0 needs to send packets to the Stargate $D$ at the other end (i.e., column 14). Using the data on unicast MAC latencies in the case of *interferer-free*, we show in Figure 3.10 the mean unicast MAC latencies and the corresponding ELD's regarding neighbors at dif-

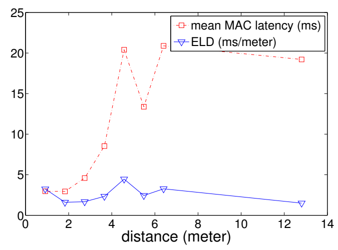


Figure 3.10: Mean unicast MAC latency and the ELD

ferent distances. From the figure, Stargate $D$, the destination which is 12.8 meters away from $S$, offers the lowest ELD, and $S$ sends packets directly to $D$. From this example, we see that, using metric ELD, a node tends to choose nodes beyond the reliable communication range as forwarders, to reduce end-to-end MAC latency as well as energy consumption.

**Remark.** ELD is a locally measurable metric based only on the geographic locations of nodes and information regarding the links associated with the sender $S$; ELD does not assume link conditions beyond the local neighborhood of $S$. In the analysis of geographic routing [83], however, a common assumption is *geographic uniformity* — that the hops in any route have similar properties such as geographic length and link quality. As we will show by experiments in Section 3.5, this assumption is usually invalid. For the sake of verification and comparison, we derive another routing metric ELR, the *expected MAC latency along a route*, based on this assumption. More specifically, $ELR(S, R) =$

$$\begin{cases} E(D_{S,R}) \times \lceil \frac{L_{S,R}+L_{R,D}}{L_{S,R}} \rceil & \text{if } L_{S,D} > L_{R,D} \\ \infty & \text{otherwise} \end{cases} \tag{3.3}$$

where $\lceil \frac{L_{S,R}+L_{R,D}}{L_{S,R}} \rceil$ denotes the number of hops to the destination, assuming equal geographic distance at every hop. We will show in Section 3.5 that ELR is inferior to ELD.

## 3.3.2 Sample size analysis

To understand the convergence speed of ELD-based routing and to guide protocol design, we experimentally study the sample size required to distinguish out the best neighbor in routing.

In our indoor testbed, let the Stargate at column 0 of row 6 be the sender $S$ and Stargate at the other end of row 6 be the destination $D$; then let $S$ send 30,000 1200-byte unicast packets to each of the other Stargates in the testbed, to get information (e.g., MAC latency and reliability) on all the links associated with $S$. The objective is to see what sample size is required for $S$ to distinguish out the best neighbor.

First, we need to derive the *distribution model* for MAC latency. Figure 3.11 shows the

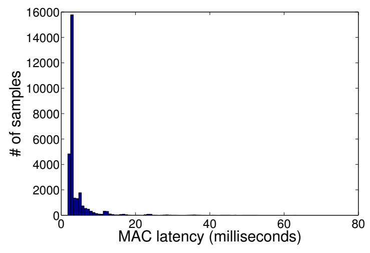


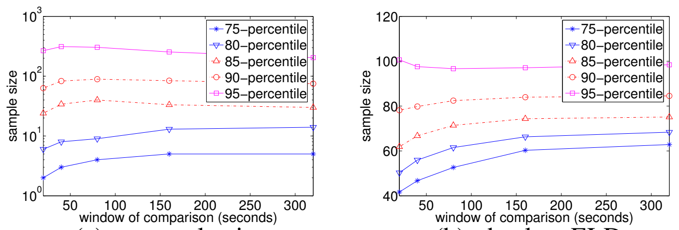
Figure 3.11: Histogram for unicast MAC latency

histogram of the unicast MAC latencies for the link to a node 3.65 meters (i.e., 12 feet) away from $S$. (The MAC latencies for other links assume similar patterns.) Given the shape of the histogram and the fact that MAC latency is a type of "service time", we select three models for evaluation: exponential, gamma, and lognormal.[5] Against the data on the MAC latencies for all the links associated with $S$, we perform Kolmogorov-Smirnov test [45] on the three models, and we find that *lognormal* distribution fits the data the best.

Therefore, we adopt lognormal distribution for the analysis in this chapter. Given that MAC latency assumes lognormal distribution, the LD associated with a neighbor also assumes lognormal distribution, i.e., $log(LD)$ assumes normal distribution.

Because link quality varies temporally, the best neighbor for $S$ may change temporally. Therefore, we divide the 30,000 MAC latency samples of each link into chunks of time span $W_c$, denoted as the *window of comparison*, and we compare all the links via their

[5]The methodology of LOF is independent of the distribution model adopted. Therefore, LOF would still apply even if better models are found later.



(a) route selection (b) absolute ELD

Figure 3.12: Sample size requirement

corresponding sample-chunks. Given each sample chunk for the MAC latency of a link, we compute the sample mean and sample variance for the corresponding $log(LD)$, and use them as the mean and variance of the lognormal distribution. When considering the $i$-th sample chunks of all the links ($i = 1, 2, \ldots$), we find the best link according to these sample chunks, and we compute the sample size required for comparing this best link with each of the other links as follows:

> Given two normal variates $X_1, X_2$ where $X_1 \sim N(\mu_1, \delta_1^2)$ and $X_2 \sim N(\mu_2, \delta_2^2)$, the sample size required to compare $X_1$ and $X_2$ at $100(1-\alpha)\%$ confidence level is $(\frac{Z_\alpha(\delta_1+\delta_2)}{\mu_1-\mu_2})^2$ $(0 \leq \alpha \leq 1)$, with $Z_\alpha$ being the $\alpha$-quantile of a unit normal variate [51].

In the end, we have a set of sample sizes for each specific $W_c$. For a 95% confidence level comparison and route selection, Figure 3.12(a) shows the 75-, 80-, 85-, 90-, and 95-percentiles of the sample sizes for different $W_c$'s. We see that the percentiles do not change much as $W_c$ changes. Moreover, we observe that, even though the 90- and 95-percentiles tend to be large, the 75- and 80-percentiles are pretty small (e.g., being 2 and 6 respectively when $W_c$ is 20 seconds), which implies that routing decisions can converge quickly in most cases. This observation also motivates us to use initial sampling in LOF, as detailed in Section 3.4.2.

**Remarks.** In the analysis above, we did not consider the temporal patterns of link properties (which are usually unknown). Had the temporal patterns been known and used in link estimation, the sample size requirement can be even lower.

By way of contrast, we also compute the sample size required to estimate the absolute ELD value associated with each neighbor. Figure 3.12(b) shows the percentiles for a 95% confidence level estimation with an accuracy of $\pm 5\%$. We see that, even though the 90- and 95-percentiles are less than those for route selection, the 75- and 80-percentiles (e.g., being 42 and 51 respectively when $W_c$ is 20 seconds) are significantly greater than those for route selection. Therefore, when analyzing sample size requirement for routing, we should focus on relative comparison among neighbors rather than on estimating the absolute value, unlike what has been done in the literature [95].

## 3.4 LOF: a data-driven protocol

Having determined the routing metric ELD, we are ready to design protocol LOF for implementing ELD without using periodic beacons. Without loss of generality, we only consider a single destination, i.e., the base station to which every other node needs to find a route.

Briefly speaking, LOF needs to accomplish two tasks: First, to enable a node to obtain the geographic location of the base station, as well as the IDs and locations of its neighbors; Second, to enable a node to track the LD (i.e., MAC latency per unit-distance to the destination) regarding each of its neighbors. The first task is relatively simple and only requires exchanging a few control packets among neighbors in rare cases (e.g., when a node boots up); LOF accomplishes the second task using three mechanisms: initial sampling of MAC latency, adapting estimation via MAC feedback for application traffic, and probabilistically exploring alternative forwarders.

In what follows, we elaborate on the individual components of LOF. (**Due to the limitation of space, we relegate to [101] the discussion on implementation issues of LOF: reliably fetching MAC feedback, reliable transport, node mobility, and neighbor-table size control**.)

### 3.4.1 Learning where we are

LOF enables a node to learn its neighborhood and the location of the base station via the following rules:

I. **[Issue request]** Upon boot-up, a node broadcasts $M$ copies of *hello-request* packets if it is not the base station. A *hello-request* packet contains the ID and the geographic location of the issuing node. To guarantee that a requesting node is heard by its neighbors, we set $M$ as 7 in our experiments.

II. **[Answer request]** When receiving a *hello-request* packet from another node that is farther away from the base station, the base station or a node that has a path to the base station acknowledges the requesting node by broadcasting $M$ copies of *hello-reply* packets. A *hello-reply* packet contains the location of the base station as well as the ID and the location of the issuing node.

III. **[Handle announcement]** When a node $A$ hears for the first time a *hello-reply* packet from another node $B$ closer to the base station, $A$ records the ID and location of $B$ and regards $B$ as a forwarder-candidate.

IV. **[Announce presence]** When a node other than the base station finds a forwarder-candidate (and thus a path) for the first time, or when the base station boots up, it broadcasts $M$ copies of *hello-reply* packets.

To reduce potential contention, every broadcast transmission mentioned above is preceded by a randomized waiting period whose length is dependent on node distribution density in the network. Note that the above rules can be optimized in various ways. For instance, rule II can be optimized such that a node acknowledges at most one *hello-request* from another node each time the requesting node boots up. Even though we have implemented quite a few such optimizations, we skip the details here since they are not the focus of this chapter.

### 3.4.2 Initial sampling

Having learned the location of the base station as well as the locations and IDs of its neighbors, a node needs to estimate the LDs regarding its neighbors. To design the estimation mechanism, let us first check Figure 3.13, which shows the mean unicast MAC latency

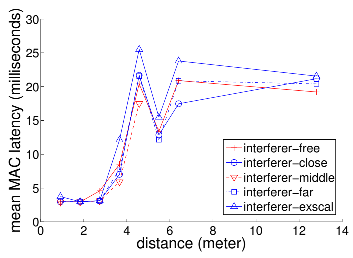


Figure 3.13: MAC latency in the presence of interference

in different interfering scenarios for the indoor experiments described in Section 3.2.1. We see that, even though MAC latencies change as interference pattern changes, the relative ranking in the mean MAC latency among links does not change much. Neither will the LDs accordingly.

In LOF, therefore, when a node $S$ learns of the existence of a neighbor $R$ for the first time, $S$ takes a few samples of the MAC latency for the link to $R$ before forwarding any data packets to $R$. The sampling is achieved by $S$ sending a few unicast packets to $R$ and then fetching the MAC feedback. The initial sampling gives a node a rough idea of the relative quality of the links to its neighbors, to jump start the data-driven estimation.

According to the analysis in Section 3.3.2, another reason for initial sampling is that, with relatively small sample size, a node could gain a decent sense of the relative goodness of its neighbors. We set the initial sample size as $6$ (i.e., the 80-percentile of the sample size when $W_c$ is 20 seconds) in our experiments.

### 3.4.3 Data-driven adaptation

Via initial sampling, a node gets a rough estimation of the relative goodness of its neighbors. To improve its route selection for an application traffic pattern, the node needs to adapt its estimation of LD via the MAC feedback for unicast data transmission. (According to the analysis in Section 3.3.2, route decisions converge quickly because of the small sample size requirement.) Since LD is lognormally distributed, LD is estimated by estimating $log(LD)$.

**On-line estimation.** To determine the estimation method, we first check the properties of the time series of $log(LD)$, considering the same scenario as discussed in Section 3.3.2. Figure 3.14 shows a time series of the $log(LD)$ regarding a node 3.65 meters (i.e., 12 feet) away from the sender $S$ (The $log(LD)$ for the other nodes assumes similar patterns.). We

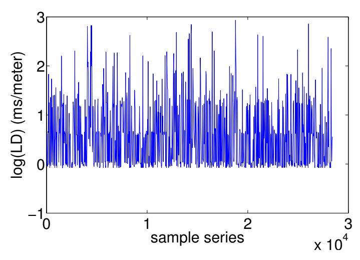


Figure 3.14: A time series of $log(LD)$

see that the time series fits well with the *constant-level model* [44] where the generating process is represented by a constant superimposed with random fluctuations. Therefore, a good estimation method is *exponentially weighted moving average* (EWMA) [44], assuming the following form

$$V \longleftarrow \alpha V + (1-\alpha)V' \tag{3.4}$$

where $V$ is the parameter to be estimated, $V'$ is the latest observation of $V$, and $\alpha$ is the weight ($0 \leq \alpha \leq 1$).

In LOF, when a new MAC latency and thus a new $log(LD)$ value with respect to the current next-hop forwarder $R$ is observed, the $V$ value in the right hand side of formula (3.4) may be quite old if $R$ has just been selected as the next-hop and some packets have been transmitted to other neighbors immediately before. To deal with this issue, we define the *age factor* $\beta(R)$ of the current next-hop forwarder $R$ as the number of packets that have been transmitted since $V$ of $R$ was last updated. Then, formula (3.4) is adapted to be the following:

$$V \longleftarrow \alpha^{\beta(R)} V + (1-\alpha^{\beta(R)})V' \tag{3.5}$$

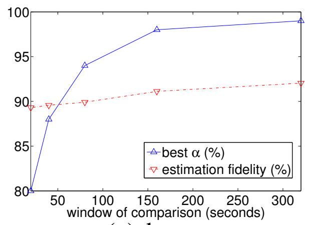
(Experiments confirm that LOF performs better with formula (3.5) than with formula (3.4).)

Each MAC feedback indicates whether a unicast transmission has succeeded and how long the MAC latency $l$ is. When a node receives a MAC feedback, it first calculates the age factor $\beta(R)$ for the current next-hop forwarder, then it adapts the estimation of $log(LD)$ as follows:

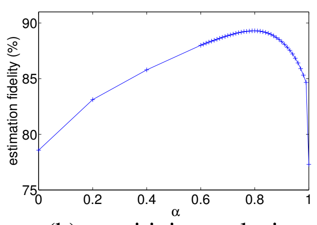
- If the transmission has succeeded, the node calculates the new $log(LD)$ value using $l$ and applies it to formula (3.5) to get a new estimation regarding the current next-hop forwarder.

- If the transmission has failed, the node should not use $l$ directly because it does not represent the latency to successfully transmit a packet. To address this issue, the node keeps track of the unicast delivery rate, which is also estimated using formula (3.5), for each associated link. Then, if the node retransmits this unicast packet via the currently used link, the expected number of retries until success is $\frac{1}{p}$, assuming that unicast failures are independent and that the unicast delivery rate along the link is $p$. Including the latency for this last failed transmission, the expected overall latency $l'$ is $(1+\frac{1}{p})l$. Therefore, the node calculates the new $log(LD)$ value using $l'$ and applies it to formula (3.5) to get a new estimation.

Another important issue in EWMA estimation is choosing the weight $\alpha$, since it affects the stability and agility of estimation. To address this question, we again perform experiment-based analysis. Using the data from Section 3.3.2, we try out different $\alpha$ values and compute the corresponding estimation fidelity, that is, the probability of LOF choosing the right next-hop forwarder for $S$. Figure 3.15(a) shows the best $\alpha$ value and the corresponding estimation fidelity for different windows of comparison. If the window of comparison is 20 seconds, for instance, the best $\alpha$ is 0.8, and the corresponding estimation fidelity is 89.3%. (Since the time span of the ExScal traffic trace is about 20 seconds, we set $\alpha$ as 0.8 in our experiments.)



(a) best $\alpha$



(b) sensitivity analysis

Figure 3.15: The weight $\alpha$ in EWMA

For sensitivity analysis, Figure 3.15(b) shows how the estimation fidelity changes with $\alpha$ when the window of comparison is 20 seconds. We see that the estimation fidelity is not very sensitive to changes in $\alpha$ over a wide range. For example, the estimation fidelity remains above 85% when $\alpha$ changes from 0.6 to 0.98. Similar patterns are observed for the other windows of comparison too. The insensitivity of estimation fidelity to $\alpha$ guarantees the robustness of EWMA estimation in different environments.

**Route adaptation.** As the estimation of LD changes, a node $S$ adapts its route selection by the ELD metric. Moreover, if the unicast reliability to a neighbor $R$ is below certain threshold (say 60%), $S$ will mark $R$ as dead and will remove $R$ from the set of forwarder-candidates. If $S$ loses all its forwarder-candidates, $S$ will first broadcast $M$ copies of *hello-withdrawal* packets and then restarts the routing process. If a node $S'$ hears a *hello-withdrawal* packet from $S$, and if $S$ is a forwarder-candidate of $S'$, $S'$ removes $S$ from its set of forwarder-candidates and update its next-hop forwarder as need be. (As a side note, we find that, on average, only 0.9863 neighbors of any node are marked as dead in both our testbed experiments and the field deployment of LOF in project ExScal [12]. Again, the withdrawing and rejoining process can be optimized, but we skip the details here.)

### 3.4.4 Exploratory neighbor sampling

Given that the initial sampling is not perfect (e.g., covering 80% instead of 100% of all the possible cases) and that wireless link quality varies temporally (e.g., from initial sampling to actual data transmission), the data-driven adaptation alone may miss using good links, simply because they were relatively bad when tested earlier and they do not get chance to be tried out later on. Therefore, we propose exploratory neighbor sampling in LOF. That is, whenever a node $S$ has consecutively transmitted $I_{ns}(R_0)$ number of data packets using a neighbor $R_0$, $S$ will switch its next-hop forwarder from $R_0$ to another neighbor $R'$ with probability $P_{ns}(R')$ (so that the link quality to $R'$ can be sampled). On the other hand, the exploratory neighbor sampling is optimistic in nature, and it should be used only for good neighbors. In LOF, exploratory neighbor sampling only considers the set of neighbors that are not marked as dead.

In what follows, we explain how to determine the sampling probability $P_{ns}(R')$ and the sampling interval $I_{ns}(R_0)$. For convenience, we consider a sender $S$, and let the neighbors of $S$ be $R_0, R_1, \ldots, R_N$ with increasing ranks.

**Sampling probability.** At the moment of neighbor sampling, a better neighbor should be chosen with higher probability. In LOF, a neighbor is chosen with the probability of the neighbor actually being the best next-hop forwarder. We derive this probability in three steps: the probability $P_b(R_i, R_j)$ of a neighbor $R_i$ being actually better than another one $R_j$, the probability $P_h(R_i)$ of a neighbor $R_i$ being actually better than all the neighbors that ranks lower than itself, and the probability $P_{ns}(R_i)$ of a neighbor $R_i$ being actually the best forwarder. Due to the limitation of space, we relegate the detailed derivation to [101].

**Sampling interval.** The frequency of neighbor sampling should depend on how good the current next-hop forwarder $R_0$ is, i.e., the sampling probability $P_{ns}(R_0)$. In LOF, we set the sampling interval $I_{ns}(R_0)$ to be proportional to $P_{ns}(R_0)$, that is,

$$I_{ns}(R_0) = C \times P_{ns}(R_0) \tag{3.6}$$

where $C$ is a constant being equal to $(N \times K)$, with $N$ being the number of active neighbors that $S$ has, and $K$ being a constant reflecting the degree of temporal variations in link quality. We set $K$ to be 20 in our experiments.

The above method of setting the sampling probability and the sampling interval ensures that better forwarders will be sampled with higher probabilities, and that the better the current forwarder is, the lower the frequency of exploratory neighbor sampling. The sampling probabilities and the sampling interval are re-calculated each time the next-hop forwarder is changed.

## 3.5 Experimental evaluation

Via testbeds and field deployment, we experimentally evaluate the design decisions and the performance of LOF. First, we present the experiment design; then we discuss the experimental results.

### 3.5.1 Experiment design

**Network setup.** In our indoor testbed as shown in Figure 3.2, we let the Stargate at the left-bottom corner of the grid be the base station, to which the other Stargates need to find routes. Then, we let the Stargate $S$ at the upper-right corner of the grid be the traffic source.

$S$ sends packets of length 1200 bytes according to the ExScal event trace as discussed in Section 3.2.1 and Figure 3.3. For each protocol we study, $S$ simulates 50 event runs, with the interval between consecutive runs being 20 seconds. Therefore, for each protocol studied, 950 (i.e., $50 \times 19$) packets are generated at $S$.

We have also tested scenarios where multiple senders generate ExScal traffic simultaneously, as well as scenarios where the data traffic is periodic; LOF has also been used in the backbone network of ExScal. We discuss them in Section 3.5.3, together with other related experiments.

**Protocols studied.** We study the performance of LOF in comparison with that of beacon-based routing, where the latest development is represented by ETX [27, 95] and PRD [83]: (For convenience, we do not differentiate the name of a routing metric and the protocol implementing it.)

- *ETX*: expected transmission count. It is a type of geography-unaware distance-vector routing where a node adopts a route with the minimum ETX value. Since the transmission rate is fixed in our experiments, ETX routing also represents another metric ETT [31], where a route with the minimum *expected transmission time* is used. ETT is similar to *MAC latency* as used in LOF.

- *PRD*: product of packet reception rate and distance traversed to the destination. Unlike ETX, PRD is geography-based. In PRD, a node selects as its next-hop forwarder the neighbor with the maximum PRD value. The design of PRD is based on the analysis that assumes geographic-uniformity.

By their original proposals, ETX and PRD use broadcast beacons in estimating the respective routing metrics. In this chapter, we compare the performance of LOF with that of ETX and PRD as originally proposed in [27] and [83], without considering the possibility of directly estimating metrics ETX and PRD via data traffic. This is because the firmware of our SMC WLAN cards does not expose information on the number of retries of a unicast transmission. (As a part of our future work, we plan to design mechanisms to estimate ETX and PRD via data traffic and study the corresponding protocol performance.) In our experiments, metrics ETX and PRD are estimated according to the method originally proposed in [27] and [83]; for instance, broadcast beacons have the same packet length and transmission rate as those of data packets. Since it has been shown that ETX and PRD perform better than protocols based on metrics such as RTT (round-trip-time) and hop-count [30, 83], we do not study those protocols here.

To verify some important design decisions of LOF, we also study different versions of LOF as follows:[6]

- *L-hop*: assumes geographic-uniformity, and thus uses metric ELR, as specified by formula (3.3), instead of ELD;
- *L-ns*: does not use the method of exploratory neighbor sampling;
- *L-sd*: considers, in exploratory neighbor sampling, the neighbors that have been marked as dead;
- *L-se*: performs exploratory neighbor sampling after every packet transmission.

For easy comparison, we have implemented all the protocols mentioned above in EmStar [2], a software environment for developing and deploying sensornets.

[6]Note: we have studied the performance of geography-unaware distance-vector routing using data-driven estimation trying to minimize the sum of MAC latency along routes, and we found that the performance is similar to that of LOF, except that more control packets are used.

**Evaluation criteria.** Reliability is one critical concern in convergecast. Using the techniques of reliable transport discussed in [101], all the protocols guarantee 100% packet delivery in our experiments. Therefore, we compare protocols in metrics other than reliability as follows:

- *End-to-end MAC latency*: the sum of the MAC latency spent at each hop of a route. This reflects not only the delivery latency but also the throughput available via a protocol [27, 31].
- *Energy efficiency*: energy spent in delivering a packet to the base station.

### 3.5.2 Experimental results

**MAC latency.** Using boxplots[7], Figure 3.16 shows the end-to-end MAC latency, in milliseconds, for each protocol. The average end-to-end MAC latency in both ETX and PRD is around 3 times that in LOF, indicating the advantage of data-driven link quality estimation. The MAC latency in LOF is also less than that of the other versions of LOF, showing the importance of using the right routing metric (including not assuming geographic uniformity) and neighbor sampling technique.

To explain the above observation, Figures 3.17, 3.18, 3.19, and 3.20 show the route hop length, per-hop MAC latency, average per-hop geographic distance, and the coefficient of

[7]Boxplot is a nice tool for describing the distribution of a data sample:

- The lower and upper lines of the "box" are the 25th and 75th percentiles of the sample. The distance between the top and bottom of the box is the interquartile range.
- The line in the middle of the box is the sample median.
- The "whiskers", lines extending above and below the box, show the extent of the rest of the sample. If there is no outlier, the top of the upper whisker is the maximum of the sample, and the bottom of the lower whisker is the minimum. An outlier is a value that is more than 1.5 times the interquartile range away from the top or bottom of the box. An outlier, if any, is represented as a plus sign.
- The notches in the box shows the 95% confidence interval for the sample median.

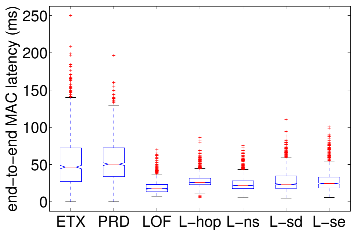


Figure 3.16: End-to-end MAC latency

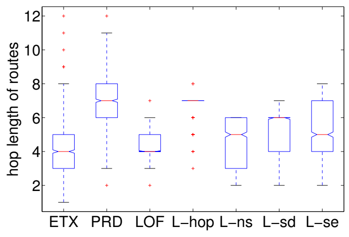


Figure 3.17: Number of hops in a route

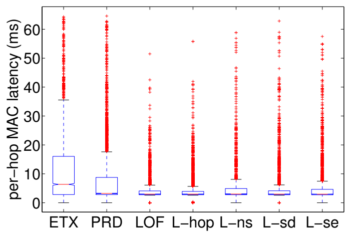


Figure 3.18: Per-hop MAC latency

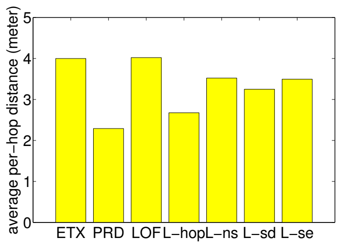


Figure 3.19: Average per-hop geographic distance

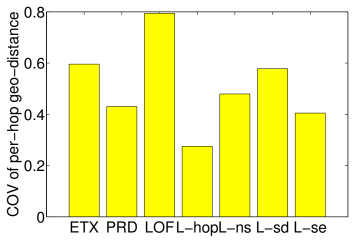


Figure 3.20: COV of per-hop geographic distance in a route

variation (COV) of per-hop geographic distance. Even though the average route hop length and per-hop geographic distance in ETX are approximately the same as those in LOF, the average per-hop MAC latency in ETX is about 3 times that in LOF, which explains why the end-to-end MAC latency in ETX is about 3 times that in LOF. In PRD, both the average route hop length and the average per-hop MAC latency is about twice that in LOF.

From Figure 3.20, we see that the COV of per-hop geographic distance is as high as 0.4305 in PRD and 0.2754 in L-hop. Therefore, the assumption of geographic uniformity is invalid, which partly explains why PRD and L-hop do not perform as well as LOF. Moreover, the fact that the COV value in LOF is the largest and that LOF performs the best tend to suggest that the network state is heterogeneous at different locations of the network.

**Energy efficiency.** Given that beacons are periodically broadcasted in ETX and PRD, and that beacons are rarely used in LOF, it is easy to see that more beacons are broadcasted in ETX and PRD than in LOF. Therefore, we focus our attention only on the number of unicast transmissions required for delivering data packets to the base station, rather than on the broadcast overhead. To this end, Figure 3.21 shows the number of unicast transmissions averaged over the number packets received at the base station. The number of unicast

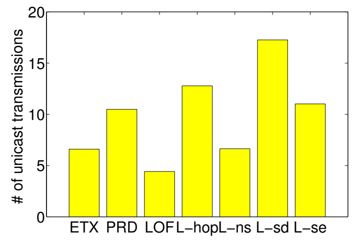


Figure 3.21: Number of unicast transmissions per packet received

transmissions per packet received in ETX and PRD is 1.49 and 2.37 times that in LOF respectively, showing again the advantage of data-driven instead of beacon-based link quality estimation. The number of unicast transmissions per packet received in LOF is also less than that in the other versions of LOF. For instance, the number of unicast transmissions in L-hop is 2.89 times that in LOF.

Given that the SMC WLAN card in our testbed uses Intersil Prism2.5 chipset which does not expose the information on the number of retries of a unicast transmission, Figure 3.21 does not represent the actual number of bytes sent. Nevertheless, given Figure 3.18 and the fact that MAC latency and energy consumption are positively related (as discussed in Section 3.3.1), the above observation on the relative energy efficiency among the protocols still holds.

To explain the above observation, Figure 3.22 shows the number of failed unicast transmissions for the 950 packets generated at the source. The number of failures in ETX and PRD is 1112 and 786 respectively, yet there are only 5 transmission failures in LOF. Also, there are 711 transmission failures in L-hop. Together with Figures 3.19 and 3.5(b), we see that there exist reliable long links, yet only LOF tends to find them well: ETX also uses long links, but they are not reliable; L-ns uses reliable links, but they are relatively shorter.

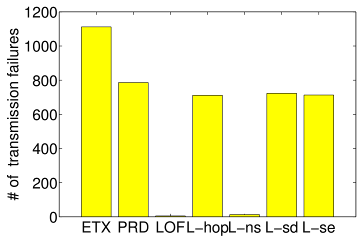
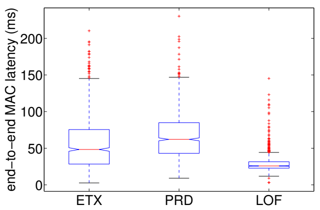


Figure 3.22: Number of failed unicast transmissions

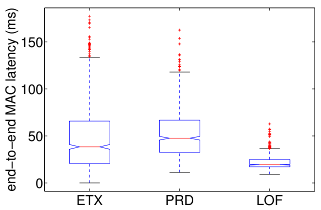
## 3.5.3 Other experiments

**Multiple senders and periodic traffic.** Besides the scenario of 1 source event traffic which we discussed in detail in the last subsection, we have performed experiments where the Stargate at the upper-right corner and its two immediate grid-neighbors simultaneously generate packets according to the ExScal traffic trace. We have also experimented with periodic traffic where 1 or 3 Stargates (same as those in the case of event traffic) generate 1,000 packets each, with each packet being 1200-byte long and the inter-packet interval being 500 milliseconds. In these experiments, we have observed similar patterns in the relative protocol performance as those in the case of 1 source event traffic. For conciseness, we only present the end-to-end MAC latency for these three cases, as shown in Figure 3.23.

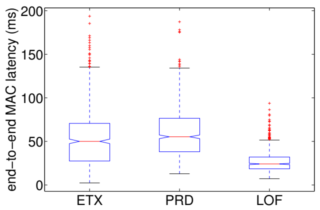
**Distance-vector data-driven routing and network throughput.** We have so far focused on geographic routing in this chapter, so that we do not need periodic control packets at all. In practice, however, it may not be feasible to have high-precision location information. In this case, we can adopt the classical distance-vector routing (which is based on the distributed Bellman-Ford algorithm) with data-driven link estimation. We have implemented



(a) event traffic, 3 senders



(b) periodic traffic, 1 sender



(c) periodic traffic, 3 senders

Figure 3.23: End-to-end MAC latency

the distance-vector data-driven routing protocol *L-dv* in EmStar, and we have experimentally measured its performance in Kansei. For the case where there is one node generating data packets according to the ExScal traffic trace, Figure 3.24 shows how the end-to-end

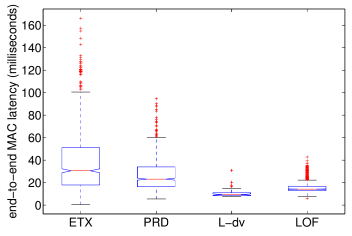


Figure 3.24: End-to-end MAC latency: event traffic, 1 sender. (Note: the experiments are done with a 15×6 subgrid of Kansei.)

MAC latency in L-dv compares with that in other protocols. We see that L-dv has similar performance as LOF, and L-dv performs better than beacon-driven routing ETX and PRD. We observe similar patterns in terms of other evaluation metrics (such as energy efficiency) and experimentation setup (such as multiple senders and periodic traffic).

To understand the impact of different protocols on network capacity, we also measure the network throughput by letting the corner source node generating data packets as fast as possible. Each data packet contains a payload of 1200 bytes. Figure 3.25 shows the

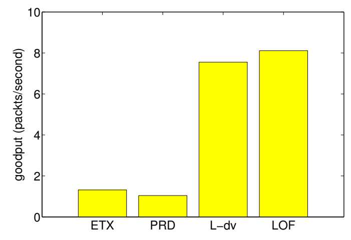


Figure 3.25: Network throughput (Note: the experiments are done with a 15×6 grid of Kansei.)

throughput in different protocols. We see that LOF and L-dv yield similar network throughput, and they both significantly improves the network throughput of ETX and PRD (e.g., up to a factor of 7.78). Our current experimental facility limits the highest achievable one-hop throughput to be  50 packets/second, thus the highest achievable multi-hop throughput is 7.14 packets/second [62]. We see that both LOF and L-dv achieve a throughput very close to the highest possible network throughput.

The data presented in Figures 3.24 and 3.25 are for experiments executed on a 15×6 grid of Kansei. When we were executing these experiments, we were unable to access the complete grid of Kansei due to some maintenance issues. But we believe the observations will carry over to other network setups including the complete grid of Kansei.

**Field deployment.** Based on its well-tested performance, LOF has been incorporated in the ExScal sensornet field experiment [12], where 203 Stargates were deployed as the backbone network, with the inter-Stargate separation being around 45 meters. LOF successfully guaranteed reliable and real-time convergecast from any number of non-base Stargates to the base station in ExScal, showing not only the performance of the protocol but also the stability of its implementation.

## 3.6 Summary

Via experiments in testbeds of 802.11b networks, we have demonstrated the difficulties of precisely estimating unicast link properties via broadcast beacons. To circumvent the difficulties, we have proposed to estimate unicast link properties via data traffic itself, using MAC feedback for data transmissions. To this end, we have modified the Linux kernel and *hostap* WLAN driver to provide feedback on the MAC latency as well as the status of every unicast transmission, and we have built system software for reliably fetching MAC feedbacks. Based on these system facilities, we have demonstrated the feasibility as well as potential benefits of data-driven routing by designing protocol LOF. LOF mainly used three techniques for link quality estimation and route selection: initial sampling, data-driven adaptation, and exploratory neighbor sampling. With its well tested performance and implementation, LOF has been successfully used to support convergecast in the backbone network of ExScal, where 203 Stargates have been deployed in an area of 1260 meters by 288 meters.

In this chapter, we have focused on data-driven link estimation and routing in 802.11 networks. But the concept of data-driven link estimation also applies to other sensornets such as those using IEEE 802.15.4 radios [104], since temporal correlation in link properties also leads to estimation inaccuracy in these networks [21]. Besides saving energy by avoiding periodic beaconing, LOF facilitates greater extent of energy conservation, because LOF does not require a node to be awake unless it is generating or forwarding data traffic. LOF also helps in enhancing network security, since the network is less exposed. More detailed study of the impact of data-driven routing on energy efficiency and security is a part of our future work.

# CHAPTER 4

# PACKING-ORIENTED SCHEDULING

In this chapter, we study the ASC (for *application-adaptive scheduling*) component of SMA from the perspective of in-network processing using packet packing (i.e., aggregating shorter packets into longer ones) as an example of in-network processing method. We first discuss the concept and benefit of packet packing in Section 4.1. In Section 4.2, we design a scheduling algorithm that improves the degree of in-network packet packing while satisfying application-specific QoS requirements. We also discuss related implementation issues In Section 4.2. We evaluate our design in Section 4.3, and we summarize this chapter in Section 4.4.

## 4.1 Packet packing

In sensornets, an information unit (e.g., a report after an event detection) from each sensor is usually short (e.g., less than 10 bytes [14]), and the header overhead of each packet is relatively high (e.g., up to 31 bytes at the MAC layer of IEEE 802.15.4). Fortunately, the maximum size of MAC payload is usually much longer than that of each information unit (e.g., 102 bytes per MAC frame in 802.15.4). Therefore, the MAC frame format allows for aggregating several short information units into a single MAC frame, which we refer to as *packet packing* hereafter. Having several information units share the overhead[8] of a packet (or frame) transmission, packet packing reduces the amortized overhead of transmitting each information unit. Packet packing also reduces the number of packets contending for channel access, hence it reduces the probability of packet collision and improves information delivery reliability, as we will show in Section 4.3.

## 4.2 Packing-oriented scheduling

Since packet packing reduces the overhead of packet transmission and channel contention, the objective of packing-oriented scheduling is to schedule packet transmissions such that as many short packets are packed into long packets as possible, by which the amortized overhead of packet transmission is reduced. To reflect the overhead of a packet transmission $tx$, we define the *amortized cost* (AC) of the transmission as $\frac{C_0}{L_{tx}}$, where $L_{tx}$ is the payload length of the packet being transmitted. Then, we can define the *utility* of a scheduling action (i.e., transmit or hold a packet) as the expected reduction in the amortized cost of packet transmissions in the network. Accordingly, whether a short packet should be held at or be immediately transmitted from a node to its parent depends on the utility of locally holding the packet and the utility of transmitting the packet.

Since locally holding a packet increases the delay in delivering the packet, the scheduling algorithm should not hold a packet too long to violate the timeliness requirement of information delivery specified by the application. Therefore, both the timeliness requirement of information delivery and application traffic pattern (e.g., spatial and temporal distributions of data packets) affect packet transmission scheduling in a network. Since the timeliness requirement and the traffic pattern vary from one application to another and

[8]The overhead includes not only the number of header bytes transmitted but also the energy taken to wake up radios, since radios may well be in low-power sleeping state in sensornets.

are usually unknown beforehand, the scheduling algorithm should adapt to the timeliness requirement and traffic pattern on the fly.

In what follows, we first discuss how to calculate the utilities of holding and transmitting a packet in an application-adaptive manner, then we present a scheduling rule that improves the overall utility.

## 4.2.1 Utility calculation

For convenience, we define the following notations:

| | | |
|---|---|---|
| $L$ | : | maximum payload length per packet; |
| $C_0$ | : | overhead of a packet transmission; |
| $ETX.j$ | : | expected number of transmissions taken to transport a packet from node $j$ to its destination; |
| $p.j$ | : | the parent or next-hop of a node $j$ in the routing tree; |
| $ETX_0.j$ | : | expected number of transmissions taken to transport a packet from node $j$ to $p.j$. |

(For simplicity of presentation, we only consider the case where every packet needs to delivered to be the base station of a sensornet [14]. The algorithm discussed in this chapter is readily applicable to the case where there are multiple base stations.)

The utilities of holding and transmitting a packet $pkt$ at a node $j$ depend on the following parameters related to traffic patterns:

- With respect to $j$ itself and its children:

| | | |
|---|---|---|
| $t_l$ | : | expected time to receive another packet $pkt'$ from a child or locally from an upper layer; |
| $s_l$ | : | expected payload size of $pkt'$. |

- With respect to the parent of $j$:

| | | |
|---|---|---|
| $t_p$ | : | expected time till the parent transmits another packet $pkt''$ that does not contain information units generated or forwarded by $j$ itself; |
| $s_p$ | : | expected payload size of $pkt''$. |

The utilities of holding and transmitting a packet $pkt$ also depend on the following constraints posed by application QoS requirement and wireless communication:

- Grace period $t_f$ for delivering $pkt$: the maximum allowable latency in delivering $pkt$ minus the expected time taken to transport $pkt$ from $j$ to its destination without being held at any intermediate node along the route.

  If $t_f \leq 0$, $pk$ should be transmitted immediately to minimize the extra delivery latency.

- Spare packet space $s_f$ of $pkt$: the maximum allowable payload length per packet minus the current payload length of $pkt$.

  Parameter $s_f$ and the size of the packets coming next from an upper layer at $j$ or from $j$'s children determine how much $pkt$ will be packed and thus the potential utility of locally holding $pkt$.

Then, the utilities of holding and transmitting a packet are calculated as follows.

**Utility of holding a packet.** When a node $j$ holds a packet $pkt$, $pkt$ can be packed with packets from $j$'s children or from an upper layer at $j$. Therefore, the utility of holding $pkt$ at $j$ is the expected reduction in the amortized cost of transmitting $pkt$ after packing $pkt$. The utility depends on (a) the expected number of packets that $j$ will receive within $t_f$ time (either from a child or locally from an upper layer), and (b) the expected payload size $s_l$ of these packets. Given that the expected inter-packet interval is $t_l$, the expected number of packets to be received at $j$ within $t_f$ time is $\frac{t_f}{t_l}$. Thus, the expected overall size $\mathcal{S}_l'$ of the payload to be received within $t_f$ time is calculated as follows:

$$\mathcal{S}_l' = \frac{t_f}{t_l} s_l$$

Given the spare space $s_f$ in the packet $pkt$, the expected size $\mathcal{S}_l$ of the payload that can be packed into $pkt$ is calculated as follows:

$$\mathcal{S}_l = \min\{\mathcal{S}_l', s_f\} = \min\{\frac{t_f}{t_l}s_l, s_f\}$$

Therefore, the expected amortized cost $AC_l$ of transporting the packet to the destination after anticipated packing is calculated as follows:

$$AC_l = \frac{C_0}{(L - s_f) + \mathcal{S}_l} ETX.j$$

where $(L - s_f)$ is the payload length of $pkt$ before packing.

Since the amortized cost $AC_l'$ of transporting $pkt$ without the anticipated packing is calculated as

$$AC_l' = \frac{C_0}{L - s_f} ETX.j$$

the utility $U_l$ of holding $pkt$ is calculated as follows:

$$\begin{aligned} U_l &= AC_l' - AC_l \\ &= \frac{C_0 \mathcal{S}_l}{(L-s_f)(L-s_f+\mathcal{S}_l)} ETX.j \\ &= \frac{C_0 \min\{\frac{t_f}{t_l}s_l, s_f\}}{(L-s_f)(L-s_f+\min\{\frac{t_f}{t_l}s_l, s_f\})} ETX.j \end{aligned} \tag{4.1}$$

**Utility of immediately transmitting a packet.** If node $j$ transmits the packet $pkt$ immediately to its parent $p.j$ when $pkt$ is not yet fully packed, $j$ pays the cost of transmitting a non-fully-packed packet. Yet the payload carried by $pkt$ can be used to pack the packets that $p.j$ has received from its children other than $j$. Therefore, the utility of $j$ transmitting a non-fully-packed packet $pkt$ comes from the expected reduction in the amortized cost of packet transmissions at $p.j$ as a result of receiving the payload that $pkt$ carries.

When $j$ transmits $pkt$ to $p.j$, the grace period of $pkt$ at $p.j$ is still $t_f$, the expected number of packets that do not contain information units from $j$ and can be packed with $pkt$ at $p.j$ is $\frac{t_f}{t_p}$. Given the limited payload that $pkt$ carries, it may happen that not all the packets to be transmitted at $p.j$ get packed (to full) via the payload from $pkt$. Accordingly, the utility $U_p$ of immediately transmitting $pkt$ is calculated as follows:

- If *all the parent packets get packed to full* via payload from $pkt$, i.e., $\frac{t_f}{t_p}(L - s_p) \leq L - s_f$:

  For each of such parent packet, the utility $U'$ (or reduction in amortized cost) is calculated as follows:

  $$\begin{aligned} U' &= \tfrac{C_0}{s_p} ETX.(p.j) - \tfrac{C_0}{L} ETX.(p.j) \\ &= \tfrac{C_0(L-s_p)}{s_p L} ETX.(p.j) \end{aligned}$$

  Then, the overall utility $U'_p$ is calculated as follows:

  $$U'_p = \frac{t_f}{t_p} U' = \frac{t_f}{t_p} \frac{C_0(L - s_p)}{s_p L} ETX.(p.j) \tag{4.2}$$

  where $\frac{t_f}{t_p}$ is the expected number of packets that do not contain information units from $j$ and can be packed with $pkt$.

- If *not all the parent packets get packed to full* via payload from $pkt$, i.e., $\frac{t_f}{t_p}(L - s_p) > L - s_f$:

  In this case, $\lfloor \frac{L-s_f}{L-s_p} \rfloor$ number of packets are packed to full, and the corresponding utility is $\lfloor \frac{L-s_f}{L-s_p} \rfloor \frac{C_0(L-s_p)}{s_p L} ETX.(p.j)$ (by Equation 4.2). In addition, there is a packet that gets partially packed via mod$(L - s_f, s_p)$ length of payload from $pkt$, and the corresponding utility is $\frac{C_0 \text{mod}(L-s_f, s_p)}{(L-s_p)(L-s_p+\text{mod}(L-s_f, s_p))} ETX.(p.j)$ (by Equation 4.1). Therefore, the overall utility $U''_p$ is the summation of the above two terms as follows:

  $$\begin{aligned} U''_p &= (\lfloor \tfrac{L-s_f}{L-s_p} \rfloor \tfrac{C_0(L-s_p)}{s_p L} + \tfrac{C_0 \text{mod}(L-s_f, s_p)}{(L-s_p)(L-s_p+\text{mod}(L-s_f, s_p))}) \times \\ &\quad ETX.(p.j) \end{aligned} \tag{4.3}$$

While immediately transmitting $pkt$ to $p.j$ brings with it the utility discussed above, immediate transmission pays the cost $C_p$ of transmitting a packet that is not full yet. According to the concept of amortized cost of packet transmission, $C_p$ is calculated as follows:

$$\begin{aligned} C_p &= \tfrac{C_0}{L-s_f} ETX_0.j - \tfrac{C_0}{L} ETX_0.j \\ &= \tfrac{C_0 s_f}{(L-s_f)L} ETX_0.j \end{aligned} \tag{4.4}$$

Therefore, the utility $U_p$ of immediately transmitting $pkt$ to $p.j$ is calculated as follows:

$$U_p = \begin{cases} U'_p - C_p & \text{if } \frac{t_f}{t_p}(L - s_p) \leq (L - s_f) \\ U''_p - C_p & \text{otherwise} \end{cases} \tag{4.5}$$

where $U'_p$, $U''_p$, and $C_p$ are defined in Equations 4.2, 4.3, and 4.4 respectively.

### 4.2.2 Scheduling rule

To reduce the amortized cost of packet transmission, the objective of packing-oriented scheduling is to maximize the utility of transmission scheduling (including the utilities of transmitting and holding packets). Since we mainly focus on demonstrating the feasibility and benefits of application-adaptivity in messaging in this chapter, we only study a greedy algorithm where each node tries to maximize the local utility of scheduling each packet transmission, and we relegate the design of globally optimal algorithm as a part of our future work.

Given a packet to be scheduled for transmission, if the probability that the packet is immediately transmitted is $P_t$ ($0 \leq P_t \leq 1$), then the expected utility $U_t(P_t)$ is calculated as follows:

$$\begin{aligned} U_t(P_t) &= P_t \times U_p + (1 - P_t)U_l \\ &= U_l + P_t(U_p - U_l) \end{aligned} \tag{4.6}$$

where $U_p$ and $U_l$ are the utilities of immediately transmitting and locally holding the packet respectively. To maximize $U_t$, $P_t$ should be set according to the following rule:

$$P_t = \begin{cases} 1 & \text{if } U_p > U_l \\ 0 & \text{otherwise} \end{cases}$$

That is, *the packet should be immediately transmitted if the utility of immediate transmission is greater than that of locally holding the packet.*

**Remarks.** The framework designed for packing-oriented scheduling is readily applicable to other in-network processing methods such as data compression, since the impact of in-network processing (no matter how it is achieved) can be modeled by the concept of *utility*. Detailed discussion of this, however, is beyond the scope of this chapter.

### 4.2.3 Implementation

From the discussion in Section 4.2.1, a node $j$ needs to obtain the following parameters when calculating the utilities of holding and transmitting a packet:

- On messaging structure: $ETX.j$, $p.j$, and $ETX_0.j$;
- On traffic pattern: $t_l$, $s_l$, $t_p$, $s_p$, and $L$.

Parameters related to messaging structure can be provided by component TLR or AST depending on the software architecture in a given system platform. On parameters related to traffic pattern, $j$ can estimate by itself the parameters $t_l$ and $s_l$, and $L$ is readily available and fixed for each specific platform. To enable each node $j$ to obtain parameters $t_p$ and $s_p$, every node $k$ in the network estimates the expected interval $t.k$ in transmitting two consecutive packets at $k$ itself and the expected size $s.k$ of these packets. Then, every node $k$ shares with its neighbors the parameters $t.k$ and $s.k$ by piggybacking these information onto data packets or other control packets in the network. When a node $j$ overhears parameter $t.(p.j)$ and $s.(p.j)$ from its parent $p.j$, $j$ can approximate $t_p$ and $s_p$ with $\frac{t.(p.j) \times t.j \times s.(p.j)}{t.j \times s.(p.j) - t.(p.j) \times s.k}$ and $s.(p.j)$ respectively. The derivation is as follows.

*Approximation of $t_p$ and $s_p$*: Since information units generated or forwarded by the children of node $p.j$ are treated in the same manner (without considering where they are from), the expected size of the packet being transmitted by $p.j$ does not depend on whether the packet contains information units generated or forwarded by $j$. Thus, $j$ can simply regard $s.(p.j)$ as $s_p$, the expected size of the packet transmitted by $p.j$ that does not contain information units coming from $j$.

Now we derive $t_p$ as follows. Since the amount of payload transmitted by $p.j$ per unit time is $\frac{1}{t.(p.j)}s.(p.j)$ and the amount of payload transmitted by $j$ is $\frac{1}{t.j}s.j$ per unit time, the amount of payload $l_p$ that are transmitted by $p.j$ but are not from $j$ per unit time is calculated as: $l_p = \frac{s.(p.j)}{t.(p.j)} - \frac{s.j}{t.j}$. Thus, the expected rate $r_p$ that $p.j$ transmits packets that do not contain information units from $j$ is calculated as: $r_p = l_p/s.(p.j) = \frac{1}{t.(p.j)} - \frac{s.j}{t.j \times s.(p.j)}$. Therefore, the expected interval $t_p$ between $p.j$ transmitting two consecutive packets that do not contain information units from $j$ is as follows: $t_p = \frac{1}{r_p} = \frac{t.(p.j) \times t.j \times s.(p.j)}{t.j \times s.(p.j) - t.(p.j) \times s.j}$.

□

## 4.3 Performance evaluation

We have implemented packing-oriented scheduling in TinyOS [5]. The implementation takes 40 bytes of RAM (plus the memory required for regular packet buffers) and 4814 bytes of ROM. To evaluate the performance of packing-oriented scheduling, we use the routing component MintRoute [95] that is readily available in TinyOS to form the routing structure. We are currently implementing the routing protocol *Learn on the Fly* (LOF) [102] in TinyOS, to provide the service for traffic-adaptive link estimation and routing (TLR) in the messaging architecture SMA. Packing-oriented scheduling is readily interoperable with LOF, but the detailed study is beyond the scope of this chapter.

To understand the benefits of application-adaptive packet packing, we implement and compare the performance of the following messaging methods:

- *noPacking*: packets are delivered without being packed in the network.

- *simplePacking*: packets are packed if they are in the same queue, but there is not packing-oriented scheduling.

- *intelliPacking*: schedule packet transmissions so that packets are packed as much as possible while satisfying application requirement on the timeliness of information delivery, i.e., employ packing-oriented scheduling as discussed in Section 4.2).

In evaluating messaging performance, we consider the case of convergecast where every information unit is transported to a singe destination — the base station which acts as the interface between a sensornet and the rest of the world. For each method, its performance is evaluated according to the following metrics:

- *Packing ratio*: the average number of information units within each transmitted packet.

- *Energy efficiency*: the number of packet transmissions and receptions required to deliver a single information unit to the base station.

- *Information delivery reliability*: the ratio of the number of unique information units received at the base station to the number of unique information units generated in the network.

In what follows, we first evaluate the performance of different messaging methods via simulation. Then we evaluate their performance via experimentation with Tmote Sky sensor nodes [6], to corroborate our observations in simulation.

### 4.3.1 Simulation study

We use the simulator TOSSIM [61] that comes with TinyOS. In the simulation, 100 nodes are deployed in a 10$\times$10 grid where each node can reliably communicate with nodes

3 grid-hops away. The traffic pattern is such that the base station is at one corner of the grid, and nodes in the farthest 4×4 subgrid from the base station periodically generate information units, with the interval between two consecutive information units uniformly distributed between 5 seconds and 15 seconds. The length of each information unit is 16 bytes, including information such as the node ID and timestamp at the source.

In the simulation, we first study a typical scenario where the maximum payload length is 102 bytes, and the application QoS requirement is specified such that the maximum allowable latency in delivering information units is 10 seconds [14, 12]. Then we study the impact of maximum allowable information delivery latency and payload length on the performance of intelliPacking.

**A typical scenario.** For the scenario where the maximum payload length is 102 bytes and the maximum allowable information delivery latency is 10 seconds, Figure 4.1 shows the

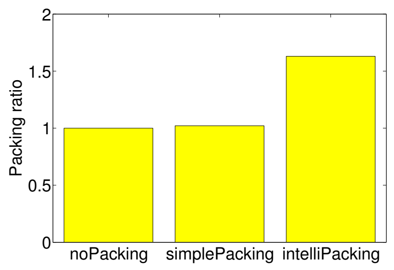
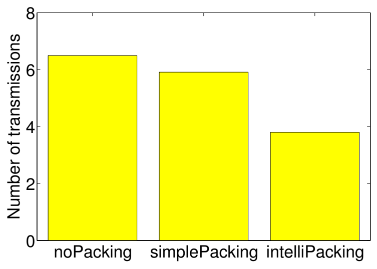


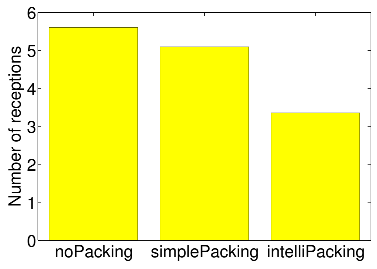
Figure 4.1: Packing ratio

packing ratio in the three messaging methods. The packing ratio is 1, 1.02, and 1.63 for noPacking, simplePacking, and intelliPacking respectively. We can see that, compared with simplePacking, intelliPacking significantly improves the packing ratio. This is because intelliPacking dynamically estimates traffic pattern and schedules packet transmissions so that the degree of in-network packet packing is improved.

As a result of the improved in-network packet packing, intelliPacking also improves energy efficiency in delivering information, as shown in Figure 4.2 Compared with noPack-



(a) Number of transmissions

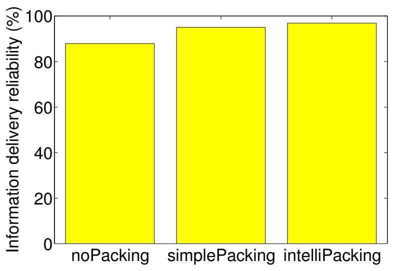


(b) Number of receptions

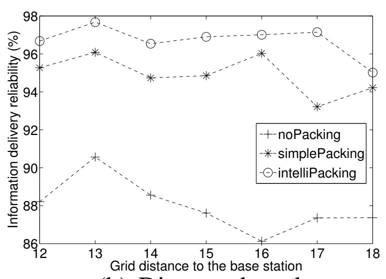
Figure 4.2: Average number of transmissions and receptions per information unit received

ing, intelliPacking reduces the average number of transmissions and receptions required for delivering an information unit by a factor of 2.33 and 2.35 respectively; compared with simplePacking, intelliPacking also reduces the average number of transmissions and receptions required for delivering an information unit by a factor of 1.59 and 1.56 respectively.

Since intelliPacking reduces the number packet transmissions, it reduces the degree of channel contention in the network and thus improves reliability in delivering information, as shown in Figure 4.3 which presents the network-wide average information delivery reliability, as well as the average reliability based on the distance (measured in grid-hops) from the source to the base station. Compared with noPacking and simplePacking, intelliPacking improves reliability by 8.98% and 1.87% respectively. (We will see a little bit later



(a) Network-wide average



(b) Distance-based

Figure 4.3: Information delivery reliability

that the improvement in information delivery reliability is even much higher in real-world hardware based experiments.)

**Impact of maximum allowable latency.** To study the impact of application properties on intelliPacking, we vary the maximum allowable latency in information delivery from 3 seconds to 25 seconds and measure the corresponding performance of intelliPacking.

Figure 4.4 shows how the packing ratio increases as the maximum allowable informa-

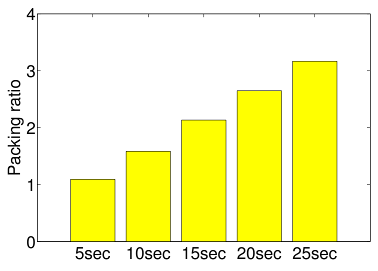
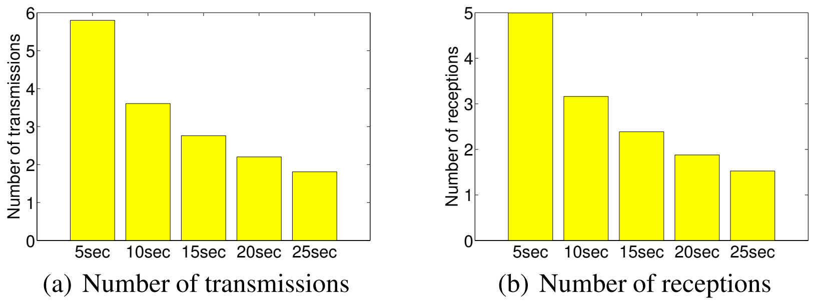


Figure 4.4: Packing ratio

tion delivery latency increases. As the allowable latency increases from 5 seconds to 25 seconds, the packing ratio increases from 1.09 to 3.17 and by a factor of 2.9.

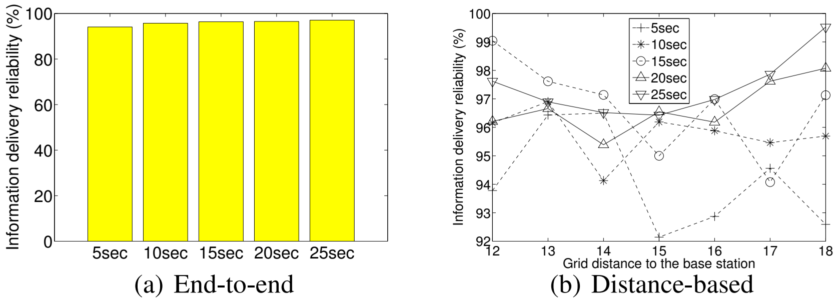
As the packing ratio increases, the energy efficiency increases by a factor up to 3.27, as shown in Figure 4.5. In the mean time, the messaging reliability increases by a factor up



(a) Number of transmissions (b) Number of receptions

Figure 4.5: Average number of transmissions and receptions per information unit received

to 3% as the channel contention decreases due to increased packing ratio. This is shown in Figure 4.6.



(a) End-to-end (b) Distance-based

Figure 4.6: Information delivery reliability

**Impact of maximum payload length.** Since the maximum packet payload length determines the maximum number of information units that can be packed into a single packet, it affects the degree of in-network packet packing. Setting the maximum information delivery latency to the typical value of 10 seconds, we vary the maximum payload length from 40 bytes to 104 bytes and measure the corresponding performance of intelliPacking. Figures 4.7, 4.8, and 4.9 present the data on packing ratio, energy efficiency, and reli-

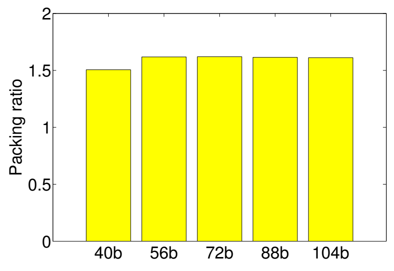
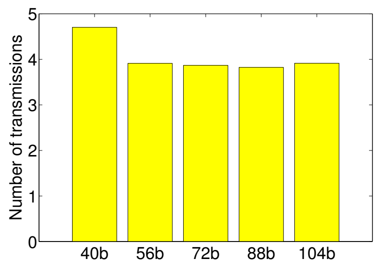
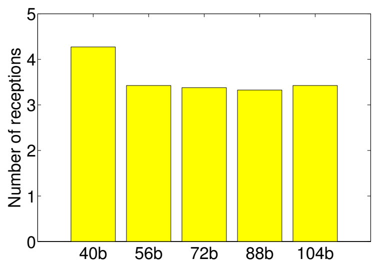


Figure 4.7: Packing ratio



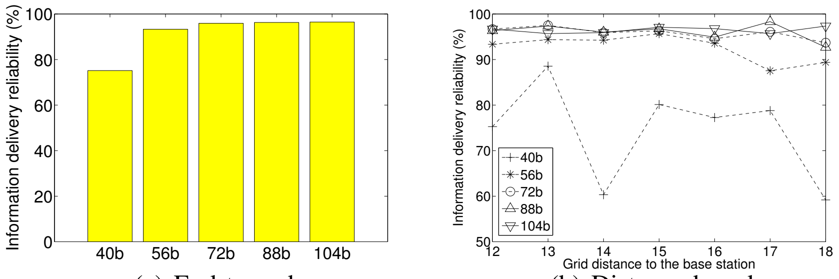
(a) Number of transmissions



(b) Number of receptions

Figure 4.8: Average number of transmissions and receptions per information unit received

ability respectively. From these figures, we see that increased maximum payload length



(a) End-to-end (b) Distance-based

Figure 4.9: Information delivery reliability

improves the overall network performance, even though the improvement is not prominent given the tight information delivery latency (which in turns bounds from above the packing ratio). Note that, when the maximum payload length is 40 bytes, each packet can carry at most 2 information units, and a packet is immediately transmitted or forwarded after each packing. As a result, compared with scenarios of longer payload length, packet transmissions are more bursty (and less spread temporally) when the maximum payload length is 40 bytes, and thus the messaging reliability is relatively lower.

Having studied the benefits and the influence factors of application-adaptive in-network packet packing in simulation, we corroborate our findings via experimentation in the next subsection.

### 4.3.2 Experimental study

To understand the performance of the different messaging methods in real-world environment, we evaluate their performance via experimentation with Tmote Sky sensor nodes. These sensor nodes use CC2420 radios which are compatible with IEEE 802.15.4 standard. We deploy 18 Tmote Sky sensor nodes in a 3$\times$6 grid with every two closest nodes separated by 1.5 feet. The sensor grid is placed in an office environment as shown in Figure 4.10. By experimenting with real-world radios and environment, we can capture the impact of

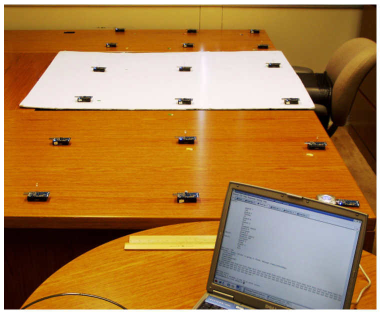


Figure 4.10: Tmote Sky sensor node grid

channel fading and channel contention, as well as the impact of temporal link properties (which did not discuss in Section 4.1).

We set the transmission power level of the sensor nodes to be 2 (out of a range from 1 to 31) such that every node can reliably communicates its immediate grid-neighbors. Similar to the typical scenario studied in simulation, the base station is at one corner of the grid, and nodes in the farthest $3\times3$ subgrid from the base station periodically generate information units, with the inter-unit interval uniformly distributed between 5 seconds and 15 seconds. The length of each information unit is 16 bytes, the maximum payload length is 102 bytes, and the maximum allowable information delivery latency is 10 seconds.

Figure 4.11 shows the packing ratio in different messaging methods. Compared with noPacking and simplePacking, intelliPacking improves the packing ratio by a factor of 5.25 and 3.5 respectively.

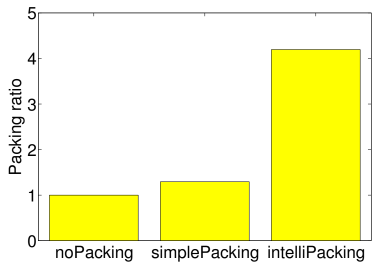
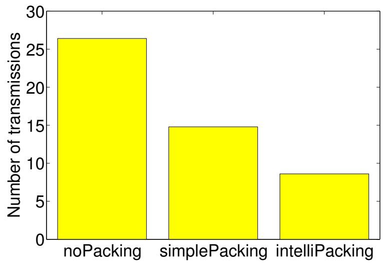
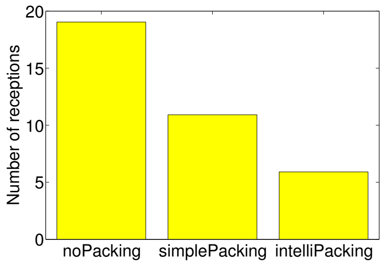


Figure 4.11: Packing ratio

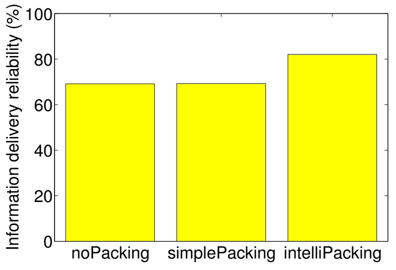


(a) Number of transmissions

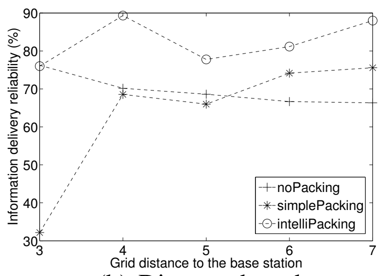


(b) Number of receptions

Figure 4.12: Average number of transmissions and receptions per information unit received



(a) End-to-end



(b) Distance-based

Figure 4.13: Information delivery reliability

Accordingly, intelliPacking significantly improves the energy efficiency, as shown in Figure 4.12. Compared with noPacking and simplePacking, intelliPacking reduces the number of transmissions required for delivering an information unit by a factor of 3.07 and 1.71 respectively, and intelliPacking reduces the number of receptions per information unit received by a factor of 3.22 and 1.85 respectively.

Because intelliPacking reduces the number of packet transmissions in the network, it reduces the degree of channel contention. Accordingly, it improves messaging reliability as shown in Figure 4.13. Compared with noPacking and simplePacking, intelliPacking improves the messaging reliability by 12.92% and 12.77% respectively.

From the above study, we see that the experiments corroborate our observations in simulation, even strengthening the observations by showing higher degree of improvement in packing ratio, energy efficiency, and information delivery reliability.

## 4.4 Summary

Taking packet packing as an example of in-network processing, we studied application-adaptive scheduling in detail. Based on the concept of *scheduling utility*, the algorithmic framework for packing-oriented scheduling is generically applicable to other in-network processing methods. Through simulation and experimentation, we have shown that our design improves both energy efficiency and reliability in sensornet messaging.

While we have application-adaptive scheduling from the perspective of packet packing, we believe our effort is only the first step toward application-adaptive scheduling in sensornets. As applications evolve, we hope to enrich our design by taking into account other in-network processing methods (e.g., information fusion) and application requirements (e.g., packet delivery reliability).

# CHAPTER 5

# RELIABLE AND REAL-TIME DATA TRANSPORT

In this chapter, we study the ASC (for *application-adaptive scheduling*) component of SMA  from the perspective of application QoS requirements. More specifically, we study transport control for event-detection applications, and we present the protocol *Reliable Bursty Convergecast* (RBC) for reliable and real-time data transport.

## 5.1 Motivation

A typical application of wireless sensor networks is to monitor an environment (be it an agricultural field or a classified area) for events that are of interest to the users. Usually, the events are rare. Yet when an event occurs, a large burst of packets is often generated that needs to be transported reliably and in real-time to a base station. One exemplary event-driven application is demonstrated in the DARPA NEST field experiment "A Line in the Sand" (simply called *Lites* hereafter) [14]. In Lites, a typical event generates up to 100 packets within a few seconds and the packets need to be transported from different network locations to a base station, over multi-hop routes.

The high-volume bursty traffic in event-driven applications poses special challenges for reliable and real-time packet delivery. The large number of packets generated within a short period leads to high degree of channel contention and thus a high probability of packet collision. The situation is further exacerbated by the fact that packets travel over multi-hop routes: first, the total number of packets competing for channel access is increased by a factor of the average hop-count of network routes; second, the probability of packet collision increases in multi-hop networks due to problems such as hidden-terminals. Consequently, packets are lost with high probability in bursty convergecast. For example, with the default radio stack of TinyOS [7], around 50% of packets are lost for most events in Lites.

For real-time packet delivery, hop-by-hop packet recovery is usually preferred over end-to-end recovery [86, 89]; and this is especially the case when 100% packet delivery is not required (for instance, for bursty convergecast in sensor networks). Nevertheless, we find issues with existing hop-by-hop control mechanisms in bursty convergecast. Via experiments with a testbed of 49 MICA2 motes and with traffic traces of Lites, we observe that the commonly used link-layer error control mechanisms do not significantly improve and can even degenerate packet delivery reliability. For example, when packets are retransmitted up to twice at each hop, the overall packet delivery ratio increases by only 6.15%; and when the number of retranmissions increases, the packet delivery ratio actually decreases, by 11.33%.

One issue with existing hop-by-hop control mechanisms is that they do not schedule packet retransmissions appropriately; as a result, retransmitted packets further increase the channel contention and cause more packet loss. Moreover, due to in-order packet delivery and conservative retransmission timers, packet delivery can be significantly delayed in existing hop-by-hop mechanisms, which leads to packet backlogging and reduction in network throughput. (We examine the details in Section 5.3.)

On the other hand, the new network and application models of bursty convergecast in sensor networks offer unique opportunities for reliable and real-time transport control:

- First, the broadcast nature of wireless channels enables a node to determine, by snooping the channel, whether its packets are received and forwarded by its neighbors.
- Second, time synchronization and the fact that data packets are timestamped relieve transport layer from the constraint of in-order packet delivery, since applications can determine the order of packets by their timestamps.

Therefore, techniques that take advantage of these opportunities and meet the challenges of reliable and real-time bursty convergecast are desired. To this end, we design the protocol *Reliable Bursty Convergecast* (RBC) for reliable and real-time data transport in event-detection applications.

In the remainder of this chapter, we describe our testbed and discuss the experiment design in Section 5.2. In Section 5.3, we study the limitations of existing hop-by-hop control mechanisms. We present the detailed design of RBC in Section 5.4, then we present the experimental results in Section 5.5. We discuss how to extend the basic design to support continuous event convergecast and to deal with queue congestion in Section 5.6. We summarize this chapter in Section 5.7.

## 5.2 Testbed and experiment design

Towards characterizing the issues in making bursty convergecast both reliable and timely, we conduct an experimental study. We choose experimentation as opposed to simulation in order to gain higher fidelity and confidence in the observations. Before presenting our study, we first describe our testbed and the experiment design.

**Testbed.** We setup our testbed to reflect the field sensor network of Lites, and we use the traffic trace for a typical event in Lites as the basis of our experiments.

The testbed consists of 49 MICA2 motes deployed in a grass field, as shown in Figure 5.1(a), where the grass is 2-4 inches tall.The 49 motes form a 7×7 grid with a 5-feet

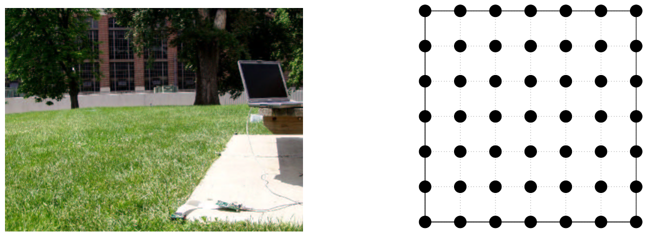

(a) Environment (b) Grid topology

Figure 5.1: The testbed

separation between neighboring grid points, as shown in Figure 5.1(b) where each grid point represents a mote. The mote at the left-bottom corner of the grid is the base station to which all the other motes send packets. The 7×7 grid imitates a subgrid in the sensor network of Lites.

The traffic trace (simply called *Lites trace* hereafter) corresponds to the packets generated in a 7×7 subgrid of the Lites network when a vehicle passes across the middle of the Lites network. When the vehicle passes by, each mote except for the base station detects the vehicle and generates two packets, which correspond to the start and the end of the event detection respectively and are separated 5-6 seconds on average. Overall, 96 packets are generated each time the vehicle passes by.[9] The cumulative distribution of the number

[9]We could have chosen a traffic trace where fewer number packets are generated (e.g., when a soldier with a gun passes by), but that would not serve as well in showing the challenges posed by huge event traffic bursts.

of packets generated during the event is shown in Figure 5.2. (Interested readers can find the detailed description of the traffic trace in [8].)

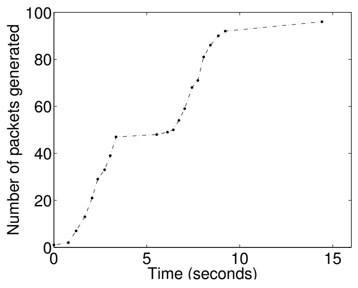


Figure 5.2: The distribution of packets generated in Lites trace

If we define the burst rate up to a moment in the event as the number of packets generated so far divided by the time since the first packet is generated, the highest burst rate in Lites trace is 14.07 packets/second. Given that the highest one-hop throughput is about 42.93 packets/second for MICA2 motes with B-MAC (the latest MAC component of TinyOS) and that, in multi-hop networks, even an ideal MAC can only achieve $\frac{1}{4}$ of the throughput that a single-hop transmission can achieve [62], the burst rate of Lites trace far exceeds the rate at which the motes can push packets to the base station. Therefore, it is a challenging task to deliver packets reliably and in real time in such a heavy-load bursty traffic scenario.

**Experiment design.** To reflect the multi-hop network of Lites, we let each mote transmit at the minimum power level by which two motes 10 feet apart are able to reliably communicate with each other, and the power level is 9 (out of a range between 1 and 255). We use the routing protocol LGR [24] in our testbed.[10] LGR uses links that are reliable in the presence of bursty traffic, and LGR spreads traffic uniformly across different paths to reduce wireless channel contention. Therefore, LGR provides a reliable and uniform packet delivery service in bursty convergecast [24]. In our testbed, the number of hops in a path is up to 6 and is 3.3 on average.

For each protocol we evaluate, we run the Lites trace 10 times and measure the average performance of the protocol by the following metrics:

- *Event reliability (ER)*: the number of unique packets received at the base station in an event divided by the number packets generated for the event.

  Event reliability reflects how well an event is reported to the base station.

- *Packet delivery delay (PD)*: the time taken for a packet to reach the base station from the node that generates it.

- *Event goodput (EG)*: the number of unique packets received at the base station divided by the interval between the moment the first packet is generated and the moment the base station receives any packet the last time.

  Event goodput reflects how fast the traffic of an event is pushed from the network to the base station. By definition, the optimal event goodput for Lites trace is 6.66 packets/second, which corresponds to the case where the packet delivery delay is 0 and all the packets are received by the base station.

- *Node reliability (NR)*: the number of unique packets that are generated by a node and received by the base station divided by the number of packets generated at the node.

[10]To focus on transport issues,we disable the "base-snooping" in LGR so that the base station does not accept packets snooped over the channel.

(Remark: The study in this chapter applies to cases where network topologies other than grid and routing protocols other than LGR are used, since the protocols studied are applicable to other network topologies and routing protocols.)

## 5.3 Limitations of two hop-by-hop packet recovery mechanisms

Two widely used hop-by-hop packet recovery mechanisms in sensor networks are synchronous explicit ack and stop-and-wait implicit ack. We study their performance in bursty convergecast as follows.

### 5.3.1 Synchronous explicit ack (SEA)

In SEA, a receiver switches to transmit-mode and sends back the acknowledgment immediately after receiving a packet; the sender immediately retransmits a packet if the corresponding ack is not received after certain constant time. Using our testbed, we study the performance of SEA when used with B-MAC [78, 7] and S-MAC [98]. B-MAC uses the mechanism of CSMA/CA (carrier sense multiple access with collision avoidance) to control channel access; S-MAC uses CSMA/CA too, but it also employs RTS-CTS handshake to reduce the impact of hidden terminals.

**SEA with B-MAC.** The event reliability, the average packet delivery delay, as well as the event goodput is shown in Table 5.1, where RT stands for the maximum number of retrans-

| Metrics | RT = 0 | RT = 1 | RT = 2 |
|---|---|---|---|
| ER (%) | 51.05 | 54.74 | 54.63 |
| PD (seconds) | 0.21 | 0.25 | 0.26 |
| EG (packets/sec) | 4.01 | 4.05 | 3.63 |

Table 5.1: SEA with B-MAC in Lites trace

missions for each packet at each hop (e.g., RT = 0 means that packets are not retransmitted). The distribution of the number of unique packets received at the base station along time is shown in Figure 5.3.

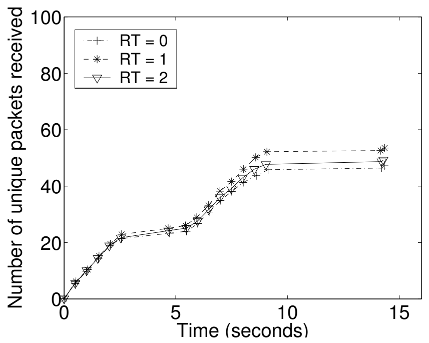


Figure 5.3: The distribution of packet reception in SEA with B-MAC

Table 5.1 and Figure 5.3 show that when packets are retransmitted, the event reliability increases slightly (i.e., by up to 3.69%). Nevertheless, the maximum reliability is still only 54.74%, and, even worse, the event reliability as well as goodput decreases when the maximum number of retransmissions increases from 1 to 2. (The above data is for B-MAC with its default contention window size. We have conducted the experiment with different contention window size of B-MAC, and we found that the performance pattern remains the same.)

**SEA with S-MAC.** Unlike B-MAC, S-MAC uses RTS-CTS handshake for unicast transmissions, which reduces packet collisions. We evaluate SEA when it is used with S-MAC, and the performance data is shown in Table 5.2 and Figure 5.4.

| Metrics | RT = 0 | RT = 1 | RT = 2 |
|---|---|---|---|
| ER (%) | 72.6 | 74.79 | 70.1 |
| PD (seconds) | 0.17 | 0.183 | 0.182 |
| EG (packets/sec) | 5.01 | 4.68 | 4.37 |

Table 5.2: SEA with S-MAC in Lites trace

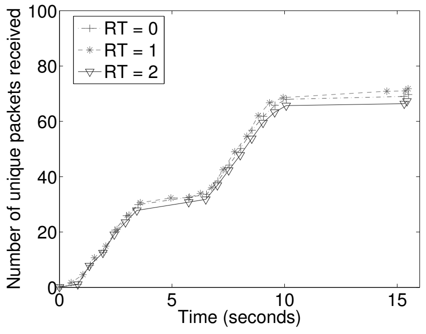


Figure 5.4: The distribution of packet reception in SEA with S-MAC

Compared with B-MAC, RTS-CTS handshake improves the event reliability by about 20% in S-MAC. Yet packet retransmissions still do not significantly improve the event reliability and can even decrease the reliability.

**Analysis.** We find that the reason why retransmission does not significantly improve — and can even degenerate — communication reliability is that, in SEA, lost packets are retransmitted while new packets are generated and forwarded, thus retransmissions, when not scheduled appropriately, only increase channel contention and cause more packet collision.[11] The situation is further exacerbated by ack-loss (with a probability as high as 10.29%), since ack-loss causes unnecessary retransmission of packets that have been received. To make retransmission effective in improving reliability, therefore, we need a retransmission scheduling mechanism that ameliorates retransmission-incurred channel contention.

### 5.3.2 Stop-and-wait implicit ack (SWIA)

SWIA takes advantage of the fact that every node, except for the base station, forwards the packet it receives and the forwarded packet can act as the acknowledgment to the sender at the previous hop [70]. In SWIA, the sender of a packet snoops the channel to check whether the packet is forwarded within certain constant threshold time; the sender regards the packet as received if it is forwarded within the threshold time, otherwise the packet is regarded as lost. The advantage of SWIA is that acknowledgment comes for free except for the limited control information piggybacked in data packets.

[11]This is not the case in wireline networks and is due to the nature of wireless communications.

We evaluate SWIA only with B-MAC, given that the implementation of S-MAC is not readily applicable for packet snooping. The performance results are shown in Table 5.3 and in Figure 5.5.

| Metrics | RT = 0 | RT = 1 | RT = 2 |
|---|---|---|---|
| ER (%) | 43.09 | 31.76 | 46.5 |
| PD (seconds) | 0.35 | 8.81 | 18.77 |
| EG (packets/sec) | 3.48 | 2.58 | 1.41 |

Table 5.3: SWIA with B-MAC in Lites trace

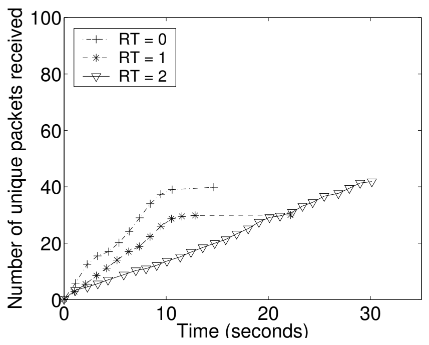


Figure 5.5: The distribution of packet reception in SWIA with B-MAC

We see that the maximum event reliability in SWIA is only 46.5%, and that the reliability decreases significantly when packets are retransmitted at most once at each hop. When packets are retransmitted up to twice at each hop, the packet delivery delay increases, and the event goodput decreases significantly despite the slightly increased reliability.

**Analysis.** We find that the above phenomena are due to the following reasons. First, the length of data packets is increased by the piggybacked control information in SWIA, thus the ack-loss probability increases (as high as 18.39% in our experiments), which in turn increases unnecessary retransmissions. Second, most packets are queued upon reception and thus their forwarding is delayed. As a result, the piggybacked acknowledgments are delayed and the corresponding packets are retransmitted unnecessarily. Third, once a packet is waiting to be acknowledged, all the packets arriving later cannot be forwarded even if the communication channel is free. Therefore, channel utilization as well as system throughput decreases, and network queuing as well as packet delivery delay increases. Fourth, as in SEA, lack of retransmission scheduling allows retransmissions, be it necessary or unnecessary, to cause more channel contention and packet loss.

## 5.4 Protocol RBC

To address the limitations of SEA and SWIA in bursty convergecast, we design protocol RBC. In RBC, we design a window-less block acknowledgment scheme to increase channel utilization and to reduce the probability of ack-loss. We also design a distributed contention control scheme that schedules packet retransmissions and reduces the contention between newly generated and retransmitted packets. Moreover, we design mechanisms to address the challenges of bursty convergecast on timer-based retransmission (such as varying ack-delay and timer-incurred delay).

Given that the number of packets competing for channel access is less in implicit-ack based schemes than in explicit-ack based schemes, we design RBC based on the paradigm of implicit-ack (i.e., piggybacking control information in data packets). We elaborate on

RBC as follows. (Even though the mechanisms used in RBC can be applied in the explicit-ack paradigm, we relegate the detailed study as our future work.)

### 5.4.1 Window-less block acknowledgment

In traditional block acknowledgment [20], a sliding-window is used for both duplicate detection and in-order packet delivery.[12] The sliding-window reduces network throughput once a packet is sent but remains unacknowledged (since the sender can only send up to its window size once a packet is unacknowledged), and in-order delivery increases packet delivery delay once a packet is lost (since the lost packet delays the delivery of every packet behind it). Therefore, the sliding-window based block acknowledgment scheme does not apply to bursty convergecast, given the real-time requirement of the latter.

To address the constraints of traditional block acknowledgment in the presence of unreliable links, we take advantage of the fact that in-order delivery is not required in bursty convergecast. Without considering the order of packet delivery, by which we only need to detect whether a sequence of packets are received without loss in the middle and whether a received packet is a duplicate of a previously received one. To this end, we design, as follows, a *window-less block acknowledgment* scheme which guarantees continuous packet forwarding irrespective of the underlying link unreliability as well as the resulting packet- and ack-loss. For clarity of presentation, we consider an arbitrary pair of nodes $S$ and $R$ where $S$ is the sender and $R$ is the receiver.

**Window-less queue management.** The sender $S$ organizes its packet queue as $(M+2)$ linked lists, as shown in Figure 5.6, where $M$ is the maximum number of retransmissions at each hop. For convenience, we call the linked lists *virtual queues*, denoted as

[12]Note that SWIA is a special type of block acknowledgment where the window size is 1.

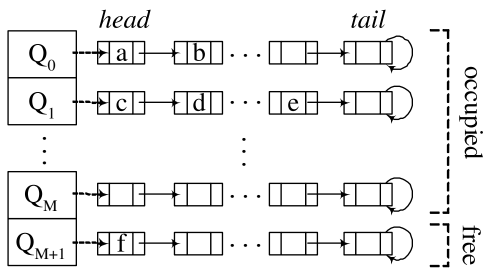


Figure 5.6: Virtual queues at a node

$Q_0, \ldots, Q_{M+1}$. The virtual queues are ranked such that a virtual queue $Q_k$ ranks higher than $Q_j$ if $k < j$.

Virtual queues $Q_0, Q_1, \ldots$, and $Q_M$ buffer packets waiting to be sent or to be acknowledged, and $Q_{M+1}$ collects the list of free queue buffers. The virtual queues are maintained as follows:

- When a new packet arrives at $S$ to be sent, $S$ detaches the head buffer of $Q_{M+1}$, if any, stores the packet into the queue buffer, and attaches the queue buffer to the tail of $Q_0$.

- Packets stored in a virtual queue $Q_k$ ($k > 0$) will not be sent unless $Q_{k-1}$ is empty; packets in the same virtual queue are sent in FIFO order.

- After a packet in a virtual queue $Q_k$ ($k \geq 0$) is sent, the corresponding queue buffer is moved to the tail of $Q_{k+1}$, unless the packet has been retransmitted $M$ times[13] in which case the queue buffer is moved to the tail of $Q_{M+1}$.

[13]Due to block-NACK, to be discussed in Section 5.4.3, a packet having been retransmitted $M$ times may be in a virtual queue other than $Q_M$.

- When a packet is acknowledged to have been received, the buffer holding the packet is released and moved to the tail of $Q_{M+1}$.

The above rules help identify the relative freshness of packets at a node (which is used in the differentiated contention control in Section 5.4.2); they also help maintain without using sliding windows the order in which unacknowledged packets have been sent, providing the basis for window-less block acknowledgment. Moreover, newly arrived packets can be sent immediately without waiting for the previously sent packets to be acknowledged, which enables continuous packet forwarding in the presence of packet- and ack-loss.

**Block acknowledgment & reduced ack-loss.** Each queue buffer at $S$ has an ID that is unique at $S$. When $S$ sends a packet to the receiver $R$, $S$ attaches the ID of the buffer holding the packet as well as the ID of the buffer holding the packet to be sent next. In Figure 5.6, for example, when $S$ sends the packet in buffer $a$, $S$ attaches the values $a$ and $b$. Given the queue maintenance procedure, if the buffer holding the packet being sent is the tail of $Q_0$ or the head of a virtual queue other than $Q_0$, $S$ also attaches the ID of the head buffer of $Q_{M+1}$, if any, since one or more new packets may arrive before the next enqueued packet is sent in which case the newly arrived packet(s) will be sent first. For example, when the packet in buffer $c$ of Figure 5.6 is sent, $S$ attaches the values $c$, $d$, and $f$.

When the receiver $R$ receives a packet $p_0$ from $S$, $R$ learns the ID $n'$ of the buffer holding the next packet to be sent by $S$. When $R$ receives a packet $p_n$ from $S$ next time, $R$ checks whether $p_n$ is from buffer $n'$ at $S$: if $p_n$ is from buffer $n'$, $R$ knows that there is no packet loss between receiving $p_0$ and $p_n$ from $S$; otherwise, $R$ detects that some packets are lost between $p_0$ and $p_n$.

For each maximal sequence of packets $p_k, \ldots, p_{k'}$ from $S$ that are received at $R$ without any loss in the middle, $R$ attaches to packet $p_{k'}$ the 2-tuple $\langle q_k, q_{k'} \rangle$, where $q_k$ and $q_{k'}$ are the IDs of the buffers storing $p_k$ and $p_{k'}$ at $S$. We call $\langle q_k, q_{k'} \rangle$ the *block acknowledgment* for packets $p_k, \ldots, p_{k'}$. When $S$ snoops the forwarded packet $p_{k'}$ later, $S$ learns that all the packets sent between $p_k$ and $p_{k'}$ have been received by $R$. Then $S$ releases the buffers holding these packets. For example, if $S$ snoops a block acknowledgment $\langle c, e \rangle$ when its queue state is as shown in Figure 5.6, $S$ knows that all the packets in buffers between $c$ and $e$ in $Q_1$ have been received, and $S$ releases buffers between $c$ and $e$, including $c$ and $e$.

One delicate detail in processing the block acknowledgment $\langle q_k, q_{k'} \rangle$ is that after releasing buffer $q_k$, $S$ will maintain a mapping $q_k \leftrightarrow q_{k''}$, where $q_{k''}$ is the buffer holding the packet sent (or to be sent next) after that in $q_{k'}$. When $S$ snoops another block acknowledgment $\langle q_k, q_n \rangle$ later, $S$ knows, by $q_k \leftrightarrow q_{k''}$, that packets sent between those in buffers $q_{k''}$ and $q_n$ have been received by $R$; then $S$ releases the buffers holding these packets, and $S$ resets the mapping to $q_k \leftrightarrow q_{n''}$, where $q_{n''}$ is the buffer holding the packet sent (or to be sent next) after that in $q_n$. $S$ maintains the mapping for $q_k$ until $S$ receives a block-NACK $[n', n)$ (to be discussed in Section 5.4.3) or a block acknowledgment $\langle q, q' \rangle$ where $q \neq q_k$, in which case $S$ maintains the mapping for $n$ or $q$ respectively. Via the buffer pointer mapped as above, the node $S$ can process the incoming block acknowledgments and block-NACKs. For convenience, we call the buffer being mapped to the *anchor* of block acknowledgments. In the examples discussed above, buffers $q_{k''}$ and $q_{n''}$ have been anchors once. We also call the packet in an anchor buffer an *anchor packet*. (The concepts of anchor and anchor packet will be used in Section 5.6.)

In the above block acknowledgment scheme, the acknowledgment for a received packet is piggybacked onto the packet itself as well as the packets that are received consecutively after the packet without any loss in the middle. Therefore, the acknowledgment is replicated and the probability for it to be lost decreases significantly, by a factor of 2.07 in Lites trace as analyzed in Appendix A.

**Duplicate detection & obsolete-ack filtering.** Since it is impossible to completely prevent ack-loss in lossy communication channels, packets whose acknowledgments are lost will be retransmitted unnecessarily. Therefore, it is necessary that duplicate packets be detected and dropped.

To enable duplicate detection, the sender $S$ maintains a counter for each queue buffer, whose value is incremented by one each time a new packet is stored in the buffer. When $S$ sends a packet, it attaches the current value of the corresponding buffer counter. For each buffer $q$ at $S$, the receiver $R$ maintains the counter value $c_q$ piggybacked in the last packet from the buffer. When $R$ receives another packet from the buffer $q$ later, $R$ checks whether the counter value piggybacked in the packet equals to $c_q$: if they are equal, $R$ knows that the packet is a duplicate and drops it; otherwise $R$ regards the packet as a new one and accepts it. The duplicate detection is local in the sense that it only requires information local to each queue buffer instead of imposing any rule involving different buffers (such as in sliding-window) that can degenerate system performance.

For the correctness of the above duplicate detection mechanism, we only need to choose the domain size $C$ for the counter value such that the probability of losing $C$ packets in succession is negligible. For example, for the high per-hop packet loss probability $22.7\%$ in the case of Lites trace, $C$ could still be as small as 7, since the probability of losing 7 packets in succession is only $0.003\%$. (Given the small domain size for the counter value as well as the usually small queue size at each node, the duplicate detection mechanism does not consume much memory. For example, it only takes 36 bytes in the case of Lites.)

In addition to duplicate detection, we also use buffer counter to filter out obsolete acknowledgment. Despite the low probability, packet forwarding at $R$ may be severely delayed, such that the queue buffers signified in a block acknowledgment have been reused by $S$ to hold packets arriving later. To deal with this, $R$ attaches to each forwarded packet the ID as well as the counter value of the buffer holding the packet at $S$ originally; when $S$ snoops a packet forwarded by $R$, $S$ checks whether the piggybacked counter value equals to the current value of the corresponding buffer: if they are equal, $S$ regards as valid the piggybacked block acknowledgment; otherwise, $S$ regards the block acknowledgment as obsolete and ignores it.

**Aggregated-ack at the base station.** In sensor networks, the base station usually forwards all the packets it receives to an external network. As a result, the children of the base station (i.e., the nodes that forward packets directly to the base station) are unable to snoop the packets the base station forwards, and the base station has to explicitly acknowledge the packets it receives. To reduce channel contention, the base station aggregates several acknowledgments, for packets received consecutively in a short period of time, into a single packet and broadcasts the packet to its children. Accordingly, the children of the base station adapt their control parameters to the way the base station handles acknowledgments.

### 5.4.2 Differentiated contention control

In wireless sensor networks where per-hop connectivity is reliable, most packet losses are due to collision in the presence of severe channel contention. To enable reliable packet delivery, lost packets need to be retransmitted. Nevertheless, packet retransmission may cause more channel contention and packet loss, thus degenerating communication reliability. Also, there exist unnecessary retransmissions due to ack-loss, which only increase channel contention and reduce communication reliability. Therefore, it is desirable to schedule packet retransmissions such that they do not interfere with transmissions of other packets.

The way the virtual queues are maintained in our window-less block acknowledgment scheme facilitates the retransmission scheduling, since packets are automatically grouped together by different virtual queues. Packets in higher-ranked virtual queues have been transmitted less number of times, and the probability that the receiver has already received the packets in higher-ranked virtual queues is lower (e.g., 0 for packets in $Q_0$). Therefore, we rank packets by the rank of the virtual queues holding the packets, and higher-ranked packets have higher-priority in accessing the communication channel. By this rule, packets that have been transmitted less number of times will be (re)transmitted earlier than those that have been transmitted more, and interference between packets of different ranks is reduced.

Window-less block acknowledgment already handles packet differentiation and scheduling within a node, thus we only need a mechanism that schedules packet transmission across different nodes. To reduce interference between packets of the same rank and to balance network queuing as well as channel contention across nodes, inter-node packet scheduling also takes into account the number of packets of a certain rank so that nodes having more such packets transmit earlier.

To implement the above concepts, we define the rank $rank(j)$ of a node $j$ as $\langle M - k, |Q_k|, \mathit{ID}(j)\rangle$, where $Q_k$ is the highest-ranked non-empty virtual queue at $j$, $|Q_k|$ is the number of packets in $Q_k$, and $\mathit{ID}(j)$ is the ID of $j$. $rank(j)$ is defined such that 1) the first field guarantees that packets having been transmitted fewer number of times will be (re)transmitted earlier, 2) the second field ensures that nodes having more packets enqueued get chances to transmit earlier, and 3) the third field is to break ties in the first two fields. A node with a larger rank value ranks higher. Then, the distributed transmission scheduling works as follows:

- Each node piggybacks its rank to the data packets it sends out.

- Upon snooping or receiving a packet, a node $j$ compares its rank with that of the packet sender $k$. $j$ will change its behavior only if $k$ ranks higher than $j$, in which case $j$ will not send any packet in the following $w(j, k) \times T_{pkt}$ time. $T_{pkt}$ is the time taken to transmit a packet at the MAC layer, and $w(j, k) = 4 - i$, when $rank(j)$ and $rank(k)$ differ at the $i$-th element of the 3-tuple ranks. $w(j, k)$ is defined such that the probability of all waiting nodes starting their transmissions simultaneously is reduced, and that higher-ranked nodes tend to wait for shorter time. $T_{pkt}$ is estimated by the method of *Exponentially Weighted Moving Average (EWMA)*.

- If a sending node $j$ detects that it will not send its next packet within $T_{pkt}$ time (i.e., when $j$ knows that, after the current packet transmission, it will rank lower than another node), $j$ signifies this by marking the packet being sent, so that the nodes overhearing the packet will skip $j$ in the contention control. (This mechanism reduces the probability of idle waiting, where the channel is free but no packet is sent.)

### 5.4.3 Timer management in window-less block acknowledgment

In window-less block acknowledgment, a sender $S$ starts a retransmission timer after sending a packet, and $S$ retransmits the packet if $S$ has not received the corresponding ack when the timer times out. Retransmission timers directly affect the reliability and the delay in packet delivery: large timeout values of timers tend to increase packet delivery delay, whereas small timeout values tend to cause unnecessary retransmissions and thus decrease packet delivery reliability. To provide reliable and real-time packet delivery, we design mechanisms to manage the timers in window-less block acknowledgment as follows. (Again, we consider a sender $S$ and a receiver $R$.)

**Dealing with varying ack-delay**

When the receiver $R$ receives a packet $m$ from the sender $S$, $R$ first buffers $m$ in $Q_0$. The delay in $R$ forwarding $m$ depends on the number of packets in front of $m$ in $Q_0$. Since the number of packets enqueued in $Q_0$ keeps changing, the delay in forwarding a received packet by $R$ keeps changing, which leads to varying delay in packet acknowledgment. Therefore, the retransmission timer at the sender $S$ should adapt to the queuing condition at $R$; otherwise, either lost packets are unnecessarily delayed in retransmission (when the retransmission timer is too large) or packets are unnecessarily retransmitted even if they are received (when the retransmission timer is too small).

To adaptively setting the retransmission timer for a packet, the sender $S$ keeps track of, by snooping packets forwarded by $R$, the length $s_r$ of $Q_0$ at $R$, the average delay $d_r$ in $R$ forwarding a packet after the packet becomes the head of $Q_0$, and the deviation $d'_r$ of $d_r$. When $S$ sends a packet to $R$, $S$ sets the retransmission timer of the packet as

$$(s_r + C_0)(d_r + 4d'_r)$$

where $C_0$ is a constant denoting the number of new packets that $R$ may have received since $S$ learned $s_r$ the last time ($C_0$ depends on the application as well as the link reliability, and $C_0$ is 3 in our experiments). The reason why we use the deviation $d'_r$ in the above formula is that $d_r$ varies a lot in wireless networks in the presence of bursty traffic, in which case the deviation improves estimation quality [49].

At a node, each local parameter $\alpha$ (such as $d_r$ for node $R$) and its deviation $\alpha'$ are estimated by the method of EWMA as follows:

$$\begin{aligned}\alpha &\leftarrow (1-\gamma)\alpha + \gamma\alpha'' \\ \alpha' &\leftarrow (1-\gamma')\alpha' + \gamma'|\alpha'' - \alpha|\end{aligned}$$

where $\gamma$ and $\gamma'$ are weight-factors, and $\alpha''$ is the latest observation of $\alpha$. Empirically, we set $\gamma = \frac{1}{8}$ and $\gamma' = \frac{1}{4}$ in RBC.

**Alleviating timer-incurred delay**

The packet retransmission timer calculated as above is conservative in the sense that it is usually greater than the actual ack-delay [49]. This is important for reducing the probability of unnecessary retransmissions, but it introduces extra delay and makes network resources under-utilized [105].

To alleviate timer-incurred delay, we design the following mechanisms to expedite necessary packet retransmissions and to improve channel utilization:

- Whenever the receiver $R$ receives a packet $m$ from buffer $n$ of the sender $S$ while $R$ is expecting (in the absence of packet loss) to receive a packet from buffer $n'$ of $S$, $R$ learns that packets sent between those in buffers $n'$ and $n$ at $S$, including the one in $n'$, are lost. In this case, $R$ piggybacks a block-NACK $[n', n)$ onto the next packet it forwards, by which the block-NACK can be snooped by $S$ immediately.

  When $S$ learns the block-NACK $[n', n)$ from $R$, $S$ resets the retransmission timers to 0 for the packets sent between those in $n'$ and $n$ (including the one in $n'$), and for each of these packets, $S$ moves the corresponding buffer to the tail of $Q_{k-1}$ if the buffer is currently at $Q_k$. Therefore, packets that need to be retransmitted are put into higher-ranked virtual queues and are retransmitted quickly.[14]

- Whenever $S$ learns that the virtual queue $Q_0$ of $R$ becomes empty, $S$ knows that $R$ has forwarded all the packets it has received. In this case, $S$ resets the retransmission timers to 0 for those packets still waiting to be acknowledged, since they will not be (due to either packet-loss or ack-loss).

  Similarly, when $S$ snoops the acknowledgment for a packet $m$, $S$ resets the retransmission timer to 0 for those packets that are sent before $m$ but are still waiting to be acknowledged.

- When a network channel is fully utilized, it should be busy all the time. Therefore, if the sender $S$ has packets to send, and if $S$ notices that no packet is sent by any neighboring node in a period of $C_1 \times T_{pkt}$ time, $S$ sends out the packet at the head of its highest-ranked non-empty virtual queue, without considering the retransmission timer even if the packet is to be acknowledged. $C_1$ is a constant reflecting the desired degree of channel utilization and $T_{pkt}$ is the time taken to transmit a packet at the MAC layer. (This mechanism improves channel utilization without introducing unnecessary retransmissions because of the "differentiated contention control" in RBC.)

## 5.5 Experimental results

We have implemented protocol RBC in TinyOS using B-MAC. In the implementation, the control logic takes 185 bytes of RAM when each node maintains a buffer capable

[14]The movement of NACKed packets affect the buffering order required by block-acknowledgment. Therefore, NACKed packets are specially marked so that they are not mistakenly regarded as having been received; the mark for a NACKed packet is reset after the packet is transmitted once more.

of holding 16 packets, and the control information piggybacked in data packets takes 14 bytes.[15] RBC has successfully provided reliable and real-time data transport in the sensor network field experiment ExScal [12] where around 1,200 Mica2/XSM motes were deployed to detect, track, and classify intruders.

We have also evaluated RBC in our testbed. In what follows, we discuss the performance of RBC, compare RBC with SEA and SWIA, discuss the impact of the individual components (i.e., window-less block acknowledgment and differentiated contention control) of RBC, and analyze the timing-shift of packet delivery in RBC.

**Performance of RBC.** Table 5.4 shows the performance results of RBC, and Figure 5.7

| Metrics | RT = 0 | RT = 1 | RT = 2 |
|---|---|---|---|
| ER (%) | 56.21 | 83.16 | 95.26 |
| PD (seconds) | 0.21 | 1.18 | 1.72 |
| EG (packets/sec) | 4.28 | 5.72 | 6.37 |

Table 5.4: RBC in Lites trace

shows the distribution of packet reception in RBC. From Table 5.4 and Figure 5.7, we observe the following properties of RBC:

- The event reliability keeps increasing, in a significant manner, as the number of retransmissions increases. The increased reliability mainly attributes to reduced unnecessary retransmissions (by reduced ack loss and adaptive retransmission timer) and retransmission scheduling.

[15]Comparatively, the control logic of SEA uses 150 bytes of RAM when each node maintains a buffer capable of holding 16 packets, and each explicit ack packet takes 16 bytes for the MICA2 radio, including the preamble, the synchronization-code, and the ack-code; the control logic of SWIA uses 68 bytes of RAM when the packet buffer size is 16, and the control information piggybacked in data packets takes 8 bytes.

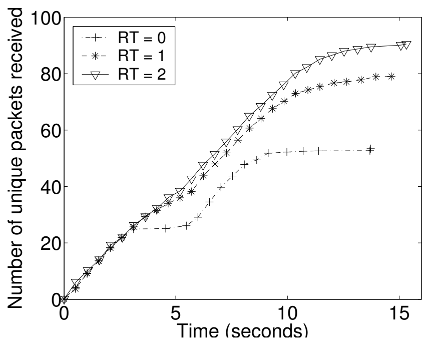


Figure 5.7: The distribution of packet reception in RBC

- Compared with SWIA which is also based on implicit-ack, RBC reduces packet delivery delay significantly. This mainly attributes to the ability of continuous packet forwarding in the presence of packet- and ack-loss and the reduction in timer-incurred delay.
- The rate of packet reception at the base station and the event goodput keep increasing as the number of retransmissions increases. When packets are retransmitted up to twice at each hop, the event goodput reaches 6.37 packets/second, quite close to the optimal goodput — 6.66 packets/second — for Lites trace.

Compared with SWIA, RBC improves reliability by a factor of 2.05 and reduces average packet delivery delay by a factor of 10.91. Compared to SEA with B-MAC (simply referred to as SEA hereafter), RBC improves reliability by a factor of 1.74, but the average packet delivery delay increases by a factor of 6.61 in RBC. Interestingly, however, RBC still improves the event goodput by a factor of 1.75 when compared with SEA. The reason is that, in RBC, lost packets are retransmitted and delivered after those packets that are generated later but transmitted less number of times. Therefore, the delivery delay for lost packets increases, which increases the average packet delivery delay, without degenerating the system goodput. The observation shows that, due to the unique application models in sensor networks, metrics evaluating aggregate system behaviors (such as the event goodput) tend to be of more relevance than metrics evaluating unit behaviors (such as the delay in delivering each individual packet).

**RBC compared with SEA and SWIA.** To further understand protocol behaviors in the presence of packet retransmissions, we conduct, as follows, a comparative study of RBC, SEA, and SWIA for the case where packets are retransmitted up to twice at each hop.

Figure 5.8 compares the distribution of packet generation in Lites trace with the distributions of packet reception in SEA, SWIA, and RBC. We see that the curve for packet reception in RBC smooths out and almost matches that of packet generation. In contrast, many packets are lost in SEA despite the fact that the rate of packet reception in SEA is close to that in RBC; packet delivery is significantly delayed in SWIA, in addition to the high degree of packet loss.

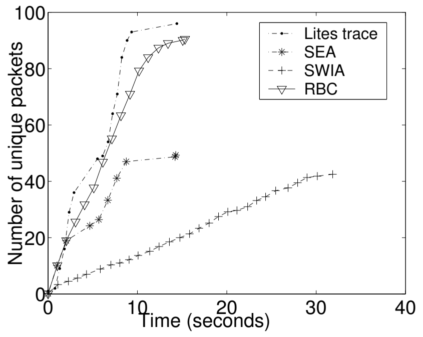
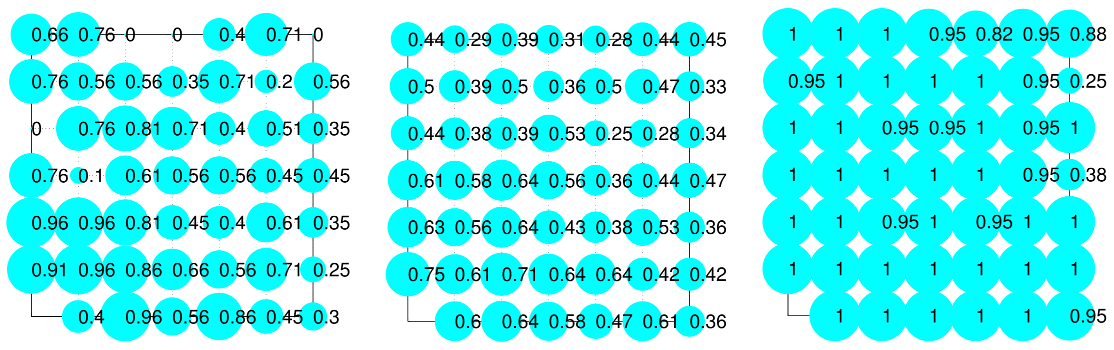


Figure 5.8: The distributions of packet generation and reception

Based on the grid topology as shown in Figure 5.1(b), Figures 5.9(a)-(c) show the node



(a) SEA (b) SWIA (c) RBC

Figure 5.9: Node reliability

reliability in SEA, SWIA, and RBC respectively. Figure 5.10 shows the cumulative distri-

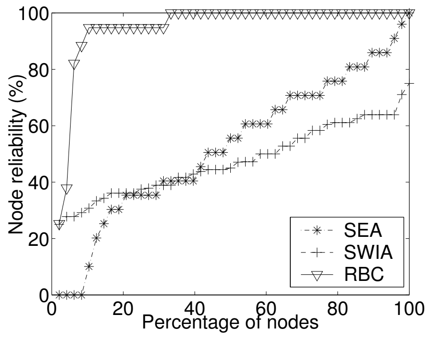


Figure 5.10: Distribution of node reliability

bution of node reliability in SEA, SWIA, and RBC. We see that node reliability improves significantly in RBC: only 4.17% of nodes have a node reliability less than 80% in RBC; yet in SEA and SWIA, above 80% of nodes have a node reliability less than 80%.

Figure 5.11 shows the average node reliability in SEA, SWIA, and RBC as the number

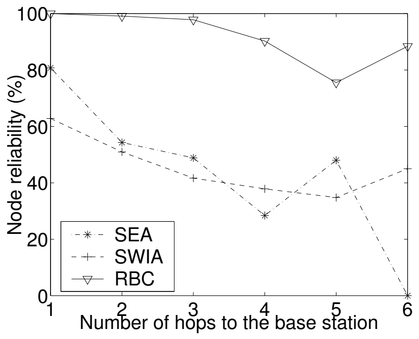


Figure 5.11: Node reliability as a function of routing hops

of routing hops (to the base station) increases. We see that the node reliability in RBC is much higher than that in SEA and SWIA at every routing hop, and that the reliability at the farthest hop in RBC is even greater than that at the closest hop in SEA and SWIA. (Note that, in RBC, the reason why nodes 5 hops away from the base station have lower average delivery rate than nodes 6 hops away can be due to the specific traffic pattern and the difference among nodes' hardware.)

**Breakdown of RBC.** To understand the individual impact of window-less block acknowledgment and differentiated contention control in RBC, we evaluate the performance of RBC without using differentiated contention control (i.e., RBC with window-less block acknowledgment only). Table 5.5 shows the performance results. Comparing Table 5.5 with

| Metrics | RT = 0 | RT = 1 | RT = 2 |
|---|---|---|---|
| ER (%) | 54.90 | 77.19 | 82.29 |
| PD (seconds) | 0.22 | 1.12 | 1.52 |
| EG (packets/sec) | 4.04 | 4.13 | 4.12 |

Table 5.5: RBC without differentiated contention control

Table 5.4, we observe the following:

- Differentiated contention control improves packet delivery performance even when there is no retransmission (i.e., RT = 0). This is because the contention control reduces channel contention by prioritizing channel access according to the degree of queue accumulation at different nodes.

- Without differentiated contention control, packet delivery reliability also improves significantly when RT (maximum number of per-hop retransmissions) increases from 0 to 1, but the improvement becomes far less when RT increases from 1 to 2. This is because differentiated contention control plays an increasingly important role when RT (thus channel contention) increases.

Comparing Table 5.5 with Tables 5.1 and 5.3, we see that, with window-less block acknowledgment alone, RBC significantly improves the packet delivery performance of SEA and SWIA. The reasons are as follows:

- Compared with SEA, the channel contention is less in window-less block acknowledgment because no explicit acknowledgment packet is generated (thus reducing the number of packets in the network). Moreover, the intra-node packet prioritization (via the queue management) in window-less block acknowledgment also improves the packet delivery reliability.

- Compared with SWIA, window-less block acknowledgment improves packet delivery reliability by reducing ack-loss probability (and thus reducing unnecessary packet retransmissions) and employing intra-node packet prioritization. Window-less block acknowledgment also significantly reduces packet delivery delay by careful timer management and by enabling continuous packet transmission in the presence of packet- and ack-loss.

**Timing-shift of packet delivery.** In this chapter, we focus on scenarios where packets are timestamped and thus we do not need to precisely preserve the relative timing between packets as it is when they are generated. Nevertheless, to characterize how RBC affects the relative timing of packets, we measure the *timing-shift* of packet delivery as follows:

> Given a packet $P1$ received at the base station, the timing-shift for $P1$ is calculated as
>
> $$|(R_1 - R_0) - (S_1 - S_0)|$$
>
> where $R_1$ denotes the time when $P1$ is received at the base station, $R_0$ denotes the time when a packet $P0$ is received at the base station immediately before $P1$, $S_1$ denotes the time when $P1$ is generated at some node in the network, and $S_0$ denotes the time when $P0$ is generated at some node in the network.[16] For convenience, we set the timing-shift to $0$ for the first packet received at the base station.

Based on the above definition, we find that the average timing-shift is 1.0035 second for the packets received in the case of RBC, and that the average timing-shift is around 0.1698 second in both SEA and SWIA. Even though the timing-shift in RBC is predictably greater than those in SEA and SWIA, it is still small enough for real-time event-driven applications such as Lites and ExScal [12] where high-level decisions are based on data in the order of seconds. (Note that, if need be, the timing-shift in packet delivery can be reduced by tuning the queue management policy in window-less block acknowledgment. But the detailed study is beyond the scope of this chapter.)

[16]In this definition, we do not consider the packets that are lost, only calculating the relative timing-shift between packets that are received at the base station.

## 5.6 Discussion

In this section, we discuss how to extend the basic design of RBC to support continuous event convergecast, and how to avoid queue overflow via flow control.

### 5.6.1 Continuous event convergecast

In event-driven applications, events are mostly rare and well-separated in time. But it may happen (though with relatively low probability) that several events happen continuously during a period of time. In what follows, we first analyze the potential issues of the basic RBC in supporting continuous event convergecast, then we extend RBC to solve the challenges.

**Starvation of orphan packets.** In RBC, window-less block acknowledgment and differentiated contention control ensures that packets having been transmitted fewer number of times will be (re)transmitted earlier. This works without any problem in the case of single event convergecast. In continuous event convergecast, however, fresh packets can be generated continuously as long as events keep occurring. Therefore, if RBC was applied without any adaptation, packets that need to be retransmitted might be delayed in transmission or never get the chance for channel access, since there might exist fresh packets at the node itself or its neighbors most of the time; that is, the packets requiring retransmission may starve in channel access.

On the other hand, not all queued packets will starve. To understand this, we divide the queued packets into two categories (according to the order of their transmission relative to the *anchor packet*[17]) as follows:

- *Orphan packets*: the packets sent earlier than the anchor packet.

[17]Defined in Section 5.4.1.

We regard these packets as orphans in the sense that their acknowledgments or NACKs have been lost. In the window-less block acknowledgment, orphan packets will not be acknowledged by the receiver any more, and they may starve if fresh packets keep arriving.

- *Awaiting packets*: the packets sent later than the anchor packet.

  We regard these packets as awaiting in the sense that they are yet to be ACKed or NACKed, but the corresponding acknowledgment packet has not been received. Given that nodes having fresh packets tend to have higher priority in channel access, and that the probability of losing several ACK or NACK packets in succession is low (e.g., 5.15% for losing 2 packets in Lites), the number of awaiting packets tend to be small. Moreover, awaiting packets will either be quickly acknowledged or quickly become orphan. If an awaiting packet is acknowledged, it will be released or moved to $Q_0$ for retransmission. Therefore, awaiting packets will not starve even if fresh packets keep arriving.

Given that only orphan packets may starve in the case of continuous event convergecast, we only need to adapt the packet scheduling of RBC to avoid starving orphan packets.

**Probabilistic scheduling of orphan packets.** One simple way to avoid starving orphan packets is to always put them in $Q_0$ once they become orphan. Nevertheless, one shortcoming of this approach is that the detection of new events may be delayed, because the delivery of new packets is delayed. This becomes undesirable especially in the case of incremental event detection where the first few packets related to an event should be delivered as soon as possible. Moreover, an orphan packet may have already been received (if the packet becomes orphan because of ack loss).

Therefore, we first analyze the importance of orphan packets in terms of the new information they may carry (since there is no need to transmit an orphan packet if it has already been received). To this end, we calculate the *probability $P_{loss}$ that an orphan packet has not been received* as follows: (Interested readers can check the detailed derivation in Appendix B.)

$$P_{loss} = \begin{cases} (1-(P'_{rbc})^k)\frac{p}{p+P'_{rbc}} & \text{if } k > 0 \\ \frac{p}{p+P'_{rbc}} & \text{if } k = 0 \end{cases}$$

where $k$ is the number of times that an orphan packet has been retransmitted due to timeout of retransmission timers, $p$ is the packet loss rate, and $P'_{rbc} = p - \frac{p(1-3p+4p^2-2p^3)}{1-p+p^2}$. In the case of Lites [8], $p = 22.7\%$. Then, $P_{loss}$ is 71.86% and 65.47% for $k = 0$ and $k = 1$ respectively. Therefore, the probability that an orphan packet has been lost in previous transmissions is high (e.g., up to 71.86% in Lites).

To take into account the probability that an orphan packet carrying new information, we adapt the intra-node and inter-node packet scheduling of RBC as follows:

- *Intra-node scheduling.* In short, the scheduling is adapted so that an orphan packet is regarded as a fresh packet with the probability that the packet has been lost. More specifically, the adaptation is as follows:
  - If the head packet $pkt_0$ of the highest ranked non-empty virtual queue $Q_k$ ($0 \leq k \leq M$) is ready for (re)transmission,[18] the node $S$ selects the head of the orphan packets[19] to transmit with the probability $P_{loss}$ that the packet has not been received. Accordingly, $S$ selects packet $pkt_0$ to transmit with probability $(1 - P'_{loss})$.

[18]A packet is ready for transmission if it is in $Q_0$ or if the retransmission timer associated with the packet is 0.

[19]The orphan packets are organized as an ordered list via the window-less queue management. Moreover, by the way RBC operates, the retransmission timers for orphan packets are 0.

  - If the head packet $pkt_0$ of the highest ranked non-empty virtual queue $Q_k$ ($0 \leq k \leq M$) is not ready for (re)transmission, $S$ selects the head orphan packet to transmit with probability 1.

- *Inter-node scheduling.* This relates to the distributed contention control in RBC. Similar to intra-node scheduling, the definition of the node rank is adapted to regard an orphan packet as one in $Q_0$ with certain probability. More specifically, if a node $S$ has $M$ orphan packets, then, in calculating its node rank, $S$ regards these orphan packets as $(\sum_{i=1}^{M} P_i)$ number of packets in $Q_0$, where $P_i$ denotes the probability that the $i$-th orphan packet has not been received.

Via the above adaptation to intra-node scheduling, we have

**Theorem 1 (Freedom of packet accumulation)** *The orphan packets at a node do not accumulate indefinitely, as long as the packet loss rate along a link is less than 49.14%.*

(Interested readers can check Appendix C for the proof.)

For routing in wireless sensor networks, reliable links are usually chosen (especially for heavy-load bursty convergecast). In Lites, for instance, the average per-hop packet delivery rate is still as high as 77.3 % despite the high channel contention. Therefore, it is reasonable to assume that routing links do have packet delivery rate greater than 50.86% (i.e., (100 - 49.14)%) in practice. Thus, the adpated intra-node scheduling not only prevents orphan packets from starving but also guarantees that orphan packets do not accumulate indefinitely. (In the worst case when some routing links happen to have reliability lower than 50.86%, the mechanisms to be discussed in the next subsection can deal with the problem of queue accumulation.)

We have implemented the above extensions to RBC, and experimentally evaluated the performance of the extended RBC in our testbed by injecting Lites traffic trace in succession. We observe the following:

- The performance (e.g., in event reliability and packet delivery delay) of the extended RBC in continuous event convergecast is very similar to that of the basic RBC in single event convergecast. Moreover, there are very few orphan packets (no more than 1 per node on average) and they do not accumulate.

- In the case of single event convergecast, the extension does not degenerate the performance of RBC. For instance, the difference between the extended and the basic version of RBC in average event reliability is within 2%.

### 5.6.2 Flow control

In the presence of high traffic load, the packet queue at a node may accumulate and overflow if the corresponding senders transmit too many packets in a short time. This issue can be avoided by proper flow control, and has been well studied in [90, 47]. We have implemented a simple hop-by-hop flow control mechanism (to work with RBC) as follows:

- When forwarding packets, a node piggybacks the number of free queue buffers at its place.

- Whenever a sender $S$ detects that the number $L_r$ of free queue buffers at the receiver $R$ is below a threshold $L$, $S$ will stop sending any packet in the following $(L - L_r) \times d_{e,R}$ time. $L$ is a constant chosen such that the probability of losing $L$ packets in succession is negligible (by which the sender will not fail to detect the congestion state at the receiver), and $d_{e,R}$ is the average interval between $R$ releasing one buffer and the next one while there are packets enqueued at $R$. ($R$ estimates $d_{e,R}$ by the method of EWMA.)

- After learning the number $L_r$ of free buffers at the receiver $R$ each time, the sender $S$ will send at most $L_r$ packets to $R$ in the following $L_r \times d_{e,R}$ time unless $S$ snoops another packet forwarded by $R$.

- To help relieve queue congestion, the nodes having less than $L$ queue buffers are not subject to the differentiated contention control.

Via testbed-based experiments and outdoor deployment in ExScal [12], the above mechanism has been proved to be highly effective in avoiding queue overflow.

## 5.7 Summary

Unlike most existing literature on reliable transport in sensor networks that focuses on periodic traffic, we have focused on bursty convergecast where the key challenges are reliable and real-time error control and the resulting contention control. To address the unique challenges, we have proposed the window-less block acknowledgment scheme which improves channel utilization and reduces ack-loss as well as packet delivery delay; we have also designed mechanisms to schedule packet retransmissions and to reduce timer-incurred delay, which are critical for reliable and real-time transport of bursty traffic. With its well-tested support for reliable and real-time transport of bursty traffic, RBC has been applied in the sensor network field experiment ExScal [12] where about 1,200 Mica2/XSM motes were deployed.

From protocol RBC, we see that bursty convergecast not only poses challenges for reliable and real-time transport control, it also provides unique opportunities for protocol design. Tolerance of out-of-order packet delivery enables the window-less block acknowledgment, which not only guarantees continuous packet delivery in the presence of packet- and ack-loss but also facilitates retransmission scheduling. Overall, the unique network as well as application models in sensor networks offer opportunities for new methodologies in protocol engineering and are interesting areas for further exploration.

In designing RBC, we have focused on reliable bursty convergecast in event-driven applications. Nevertheless, we believe some techniques of RBC (e.g., differentiated contention control) can be applied to the case where data traffic is periodic or continuous. Detailed study of this is beyond the scope of this chapter, and we regard it as a part of the future work.

# CHAPTER 6

# LOCALLY-STABILIZING SHORTEST PATH ROUTING

Having focused on addressing the challenges of wireless communication, resource constraints, and application diversity in the previous four chapters, we now turn our attention to studying the challenges of complex faults and large system scale in sensornets. More specifically, we study the modeling and algorithmic design issues related to scalable dependability in the presence of complex faults and large system scale.

## 6.1 Motivation

A well-known ideal in networking is the ability to withstand failure or compromise of one or more regions in a network without impacting a large part of the network. Yet, in many instances, we find that even a small fault-perturbed region impacts a large part of the network, as the effects of the faults propagate to and contaminate far away nodes. An example is inter-domain routing in the Internet by the Border Gateway Protocol (BGP), where faults at some edge routers can propagate across the whole Internet [59, 92].

Unbounded fault propagation decreases not only the availability of a network but also its stability and scalability. Therefore, in large-scale networks such as the Internet and the emerging wireless sensor networks [73, 92, 99], it is desirable that faults be contained locally around the regions where they have occurred, and that the time taken for a system to stabilize is a function $\mathcal{F}$ of the size of the fault-perturbed regions instead of the size of the system. We call this property *$\mathcal{F}$-local stabilization*.

**Local stabilization in routing.** One problem where $\mathcal{F}$-local stabilization is critical but remains unsolved is the basic problem of shortest path routing in networks. Generally speaking, there are two categories of routing protocols: link-state and distance-vector. In link-state protocols, each node maintains the topological information of a whole network, and $\mathcal{F}$-local stabilization is impossible, since every single change in the network topology has to be propagated to every node in the network. In distance-vector protocols, each node only maintains the distance of and the next-hop on its shortest path to each destination in the network. Thus, $\mathcal{F}$-local stabilization is conceivable in distance-vector protocols.

Distance-vector (and its variant, path-vector) protocols for the Internet, such as Routing Information Protocol (RIP) and BGP, have long been studied [46]. Distance-vector protocols for mobile ad hoc networks, such as Destination Sequenced Distance-Vector (DSDV) and Ad hoc On-Demand Distance-Vector (AODV), have also been proposed [74]. In designing these protocols, researchers have typically concentrated on how to avoid routing loops and the count-to-infinity problem. Local stabilization is not guaranteed: small-scale local perturbations (such as memory overflow) can propagate globally across a whole network, due to the diffusing nature of these protocols [92], and result in severe instability [58, 92]. Moreover, the fault model has been typically limited to node and link faults such as crash, repair, and congestion; state corruption is not considered. However, several kinds of state corruption do arise as a result of misconfiguration and faulty software, and are known to be major causes for routing instability [59, 84, 92]. And theoretically speaking, even simple faults such as node crash and message loss, can drive a network into arbitrary states [52]. Therefore, $\mathcal{F}$-local stabilization is desirable, and not only in the presence of node/link crash, repair, and congestion but also in the presence of state corruption.

In the remainder of this chapter, we present the system, fault, and computation model in Section 6.2. In Section 6.3, we define local stabilization, and analyze the properties of locally stabilizing systems. We present our LSRP protocol that solves the problem of local stabilization in shortest path routing in Section 6.4, and analyze its properties in Section 6.5. In Section 6.6, we discuss the impact of network topology on local stabilization. We summarize the chapter in Section 6.7.

## 6.2 Preliminaries

In this section, we present the system model, protocol notation, fault model, and computation model adopted in our work.

**System model.** A system $G$ is a connected undirected graph $(V, E, W)$, where $V$ and $E$ are the set of nodes and the set of edges in the system respectively, and $W$ is a positive function that defines the weight of each edge in $E$. ($W$ is also called the *weight function* hereafter.) Each node in the system has a unique $ID$. If nodes $i$ and $j$ can communicate with each other directly, then edge $(i, j)$ is in $E$. For each edge $(i, j) \in E$, its weight is denoted by $w.i.j$.

There is a clock at each node. The ratio of clock speeds between any two neighboring nodes in the system is bounded from above by $\alpha$, but no extra constraint on the absolute values of clocks is enforced.

Message transmission between nodes is reliable, and message passing delay along an edge is bounded from above and from below by $U$ and $L$ respectively.

**Protocol notation.** We write protocols using a variant of the Abstract Protocol notation [41]. At each node, the protocol consists of a finite set of variables and actions. Each action consists of three parts: guard, guard hold-time, and statement. For convenience, we associate a unique name with each action. Thus, an action has the following form:

$$\langle name \rangle :: \langle guard \rangle \xrightarrow{\quad d \quad} \langle statement \rangle$$

The guard is either a boolean expression over the protocol variables of the node or a message reception operation, $d$ is the guard hold-time ($d \geq 0$), and the statement updates zero or more protocol variables of the node and/or sends out some message(s). If $d = 0$, we write the action in the following form:

$$\langle name \rangle :: \langle guard \rangle \longrightarrow \langle statement \rangle$$

For an action whose guard is a message reception operation, its guard hold-time must be $0$.

For an action named $a$, its guard hold-time is denoted by $d.a$. An action $a$ is enabled at time $t$ if the guard of $a$ evaluates to true at $t$. An action $a$ is executed at time $t$ only if $a$ is continuously enabled from time $(t - d.a)$ to $t$. To execute an action, its statement is executed atomically.

**Fault model.** A node or an edge is *up* if it functions correctly, and it is *down* if it fail-stops. In a system, nodes and edges that are up can fail-stop, nodes and edges that are down can become up and join the system, the state of a node, i.e., the values of all the variables of the node, can be corrupted, and the weight function can change.

The protocol actions of a node cannot be corrupted.

**Computation model.** The topology of a system $G$ is the subgraph $G'(V', E')$ of $G(V, E)$ such that $V' = \{i : i \in V \wedge \ i \text{ is up}\}$ and $E' = \{(i, j) : i \in V' \wedge j \in V' \wedge (i, j) \in$

$E \wedge (i, j)$ is up$\}$. Due to faults, the system topology $G'(V', E')$ may change in the sense that the set of up nodes $V'$ or the set of up edges $E'$ changes over time. For example, node $i$ is removed from $V'$ when node $i$ fail-stops. To reflect changes in system topology as well as weight function, we regard the state of $G$ as the union of the current system topology, the current weight function, the state of all the up nodes, and the message(s) in the up edges (i.e., the messages that are sent but not yet received). At a system state $q$, the system topology, the weight function, and the state of an up node $i$ are denoted as $G.q(V.q, E.q)$, $W.q$, and $q(i)$ respectively. Given a system topology $G.q(V.q, E.q)$ and a problem specification, there exist a set of legitimate system states, denoted as $Q_l(G.q)$.

A system computation $\beta$ is either a finite sequence $q_0$, $(a_1, t_1)$, $q_1$, $(a_2, t_2)$, $\ldots$, $q_n$, or an infinite sequence $q_0$, $(a_1, t_1)$, $q_1$, $(a_2, t_2)$, $\ldots$, $q_{r-1}$, $(a_r, t_r)$, $q_r$, $\ldots$, of alternating system states (i.e. $q_0, q_1, \ldots$) and protocol actions (i.e. $a_1, a_2, \ldots$), where (i) for every $k \geq 1$, $t_k \leq t_{k+1}$, and each state transition $q_{k-1}, (a_k, t_k), q_k$ means that the execution of action $a_k$ at time $t_k$ changes the system state from $q_{k-1}$ to $q_k$; and (ii) for any two pairs $(a_k, t_k)$ and $(a_{k'}, t_{k'})$ in $\beta$ ($k \neq k'$), if $a_k$ and $a_{k'}$ are actions of the same node, then $t_k \neq t_{k'}$ (i.e., at most one action can be executed at a node at any time). $\beta$ is a finite sequence only if it ends with a state $q_n$, and there is no enabled action at $q_n$. A subsequence $\gamma$ of $\beta$ is called a computation segment if $\gamma$ starts and ends with a state.

A system computation $\beta$ can also be regarded as a sequence of *round*s. A round is a minimal computation segment $\gamma$ that starts at a state $q_k$ ($k \geq 0$) and, in $\gamma$, (i) every up node that has an action $a$ continuously enabled from some time $t'$ to $(t' + d.a)$, where $t' \leq t_k$ and $(t' + d.a) \geq t_k$, executes at least one action, and (ii) if a message is sent to a node $i$, the action that receives the message must be executed at $i$. (We assume $t_0$ is the time when $\beta$ starts.)

## 6.3 Local stabilization: concepts and properties

In this section, we first define concepts related to local stabilization, which are generic for networking and distributed computing problems, and then we present some notable properties of $\mathcal{F}$-local stabilizing systems.

### 6.3.1 Concepts related to local stabilization

In a distributed system, the variables that each node needs to maintain depend both on the problem and on the protocol being used; some are inherent in the problem itself and independent of the protocol being used, while others are dependent on the protocol. In the problem of shortest path routing, for instance, every node has to maintain the distance and the next-hop on its chosen shortest path to each destination: maintaining the distance is necessary for a node to coordinate with others to find its shortest path to the destination, and maintaining the next-hop is necessary for a node to forward packets to the destination. Therefore, the variables used to record the distance and the next-hop are inherent in the problem of shortest path routing. We call variables that are inherent in the problem *problem-specific variables*. At a system state $q$, the value of the set of problem-specific variables at a node $i$ is denoted as $q(i.p)$.

**Dependency among nodes & edges.** Given a problem, a node may depend on another node or edge in a distributed system, because, when faults occur to the latter, the former may have to change the values of its problem-specific variables in order for the system to converge to a legitimate state (i.e., to stabilize), no matter which protocol is used. For example, in the problem of shortest path routing, every node whose only shortest path to a destination goes through a node $i$ or an edge $e$ depends on $i$ or $e$ because it would have to change the next-hop on its shortest path to the destination if $i$ or $e$ fail-stopped.

In general, if some up nodes in a system have to adapt the values of their problem-specific variables in order for the system to stabilize (irrespective of the state to which the system stabilizes) after a set of nodes and edges fail-stop while the system is at a legitimate state, we regard these up nodes as dependent upon the fail-stopped nodes and edges. Similarly, if some existing nodes in a system have to adapt the values of their problem-specific variables after a set of nodes and edges newly join the system while it is at a legitimate state, we regard these existing nodes as dependent upon the newly-joining nodes and edges; for convenience, we also regard the newly-joining nodes as dependent on themselves, since they need to adapt the values of their problem-specific variables too.

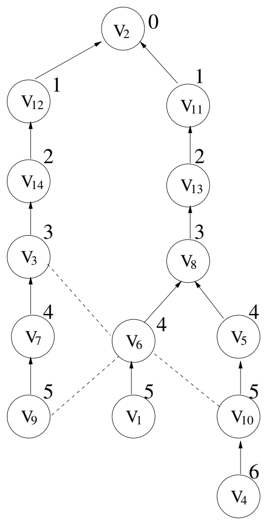
Formally, given a set of nodes $V'$, a set of edges $E'$, and a legitimate state $q$, we define the *dependent set of* $V'$ *and* $E'$ *at* $q$, denoted by $D_q(V', E')$, as:

$$\begin{cases} \{k : k \in V.q \land (\forall q' : q' \in Q_l(G_-) \Rightarrow q'(k.p) \neq q(k.p))\} \\ \qquad\qquad\qquad\qquad \text{if } V' \subseteq V.q \text{ and } E' \subseteq E.q; \\ \\ \{k : k \in V.q \land (\forall q' : q' \in Q_l(G_+) \Rightarrow q'(k.p) \neq q(k.p))\} \cup \\ V' \\ \qquad\qquad\qquad\qquad \text{if } V' \cap V.q = E' \cap E.q = \emptyset. \end{cases}$$

where $G_-$ is the system topology after $V'$ and $E'$ have fail-stopped, and $G_+$ is the system topology after $V'$ and $E'$ newly join the system, i.e., $G_- = (V.q \setminus V', E.q \setminus E')$, $G_+ = (V.q \cup V', E.q \cup E')$.[20] By definition, the dependent set $D_q(V', E')$ denotes the minimum set of nodes that are affected when the set of nodes $V'$ and the set of edges $E'$ fail-stop or newly join the system while it is at state $q$. (Note that the definition also applies to changes in link weight, since the weight change at a link can be regarded as the fail-stop of the link with the old weight followed by the join of the link with the new weight.)

Considering the problem of shortest path routing, for example, Figure 6.1 represents a legitimate state $q$. If node $v_{11}$ and edge $(v_2, v_{12})$ fail-stop at $q$, all the other nodes except

[20]The node set $V'$ and edge set $E'$ should be such that $G_-$ and $G_+$ are valid graphs.



In the figure, each circle represents a node in the system, the string in a circle represents the $ID$ of the node, node $v_2$ is a destination node, and the number besides each circle represents the distance from that node to $v_2$. A directed edge $< v_i, v_j >$ means that $v_j$ is the next-hop on the chosen shortest path from $v_i$ to $v_2$, and an undirected dashed edge $(v_i, v_j)$ means that $v_i$ and $v_j$ are neighbors in the system, but neither is $v_j$ on the chosen shortest path from $v_i$ to $v_2$, nor is $v_i$ on the chosen shortest path from $v_j$ to $v_2$. For clarity of presentation, we assume that the weight of each edge is 1.

Figure 6.1: A legitimate system state

for $v_2$ in the system need to invalidate their distance values as well as their next-hops on their paths to $v_2$, since there exists no route from any node to $v_2$ any more. Thus, $D_q(\{v_{11}\}, \{(v_{12}, v_2)\}) = \{v_1, v_3, \ldots, v_{10}, v_{13}, v_{14}\}$. Similarly, if the edge $(v_2, v_8)$ joins at $q$, node $v_9$ and the nodes in the subtree rooted at $v_8$ need to change their distance values.[21] Thus, $D_q(\emptyset, \{(v_2, v_8)\}) = \{v_8, v_6, v_5, v_9, v_1, v_{10}, v_4\}$.

**Perturbation size.** Therefore, a node $j$ can be affected by a fault in two ways irrespective of the protocols used: $j$ is directly affected by a state corruption which occurs to $j$ itself, and $j$ is indirectly affected by a non-state-corruption fault (such as fail-stop) which occurs to a node or an edge that $j$ depends on. Then, corresponding to each set of faults that leads a system to an illegitimate state $q$, there is a set of nodes in $V.q$ that are affected either directly or indirectly by the faults, and the number of affected nodes denotes the degree of perturbation by the faults. Given an illegitimate state $q$, it could have been reached in

[21]Note that nodes $v_8$ and $v_9$ also need to change their next-hops.

different ways (i.e., from different legitimate states by different sets of faults), thus the number of affected nodes at $q$ depends on how the system reaches $q$ by certain faults. To characterize the minimum amount of work required to recover from the perturbation at $q$, we define the *perturbation size at* $q$ as the minimum number of affected nodes at $q$ considering all the possible ways $q$ could have been reached. Formally,

**Definition 1 (Perturbation size)** *The perturbation size at a system state* $q$*, denoted as* $P(q)$*, is* $\min_{q' \in Q_l} |A_{q'} \cup B_{q'}|$ *where*

$$\begin{array}{l} Q_l \text{ is the set of all possible legitimate system states}, \\ A_{q'} = \{i : i \in (V.q \cap V.q') \wedge q(i) \neq q'(i)\}, \\ B_{q'} = \{i : i \in V.q \wedge (i \in V.q' \Rightarrow q(i) = q'(i)) \wedge \\ \qquad i \in (D_{q'}(V.q' \setminus V.q, E.q' \setminus E.q) \cup \\ \qquad\quad D_{q'}(V.q \setminus V.q', E.q \setminus E.q'))\} \end{array}$$

If $q$ is reached from a legitimate state $q'$ by some faults, $A_{q'}$ in the above definition denotes the set of nodes where state corruption occurred, and $B_{q'}$ denotes the set of nodes that depend on some other nodes where certain non-state-corruption faults occurred. Intuitively, the perturbation size at $q$ equals to the minimum number of nodes in $V.q$ whose states either have been corrupted by some transient faults or the values of whose problem-specific variables have to be changed in order for the system to stabilize. Thus, it also reflects the minimum amount of work needed to correct a perturbation.

Given an illegitimate state $q$, there may exist two legitimate states $q_1$ and $q_2$ such that $|A_{q_1} \cup B_{q_1}| = |A_{q_2} \cup B_{q_2}| = P(q)$. Consider a consensus problem where all the nodes in a system need to take the same value, for instance, at a state $q$ where one half of the nodes in the system take $3$ and the other half take $4$, there exist two legitimate states $q_1$ and $q_2$ such that every node takes $3$ at $q_1$, every node takes $4$ at $q_2$, and $|A_{q_1} \cup B_{q_1}| = |A_{q_2} \cup B_{q_2}| = P(q) = \frac{k}{2}$, where $k$ is the number of nodes in the system. To reflect the above situation, we define the *set of potentially perturbed sets of nodes at state* $q$, denoted by $PP(q)$, as $\{A_{q'} \cup B_{q'}: q' \in Q_l \wedge |A_{q'} \cup B_{q'}| = P(q)\}$.

To illustrate the concept of perturbation size for the problem of shortest path routing, let us consider scenarios when different faults occur while a system is at a legitimate state $q'$ as shown in Figure 6.1:

- If a state corruption occurs to node $v_8$, then the perturbation size at the state after the corruption is $1$ and the set of potentially perturbed set of node is $\{\{v_8\}\}$, since only $v_8$ needs to change its state in order for the system to stabilize to the legitimate state, and at least one node in the system needs to change its state in order for the system to stabilize.

- If node $v_8$ fail-stops, then the perturbation size is $3$ and the set of potentially perturbed set of nodes is $\{\{v_6, v_5, v_{10}\}\}$, since nodes $v_6$, $v_5$, and $v_{10}$ have to change their next-hop on their shortest paths to $v_2$, while all the other nodes in the system don't need to.

**Local stabilization.** Based on the protocol-independent concept of perturbation size which reflects the minimum amount of work required for a system to stabilize from a state, we define the concept of $\mathcal{F}$-local stabilization which reflects the properties of protocols in the presence of faults.

**Definition 2 ($\mathcal{F}$-local stabilization)** *A system $G$ is $\mathcal{F}$-local stabilizing if and only if*

> *Starting at an arbitrary state $q$, every computation of $G$ reaches a legitimate state within $\mathcal{F}(P(q))$ time, where $\mathcal{F}$ is a function and $P(q)$ is the perturbation size at state $q$.*

If a system is $\mathcal{F}$-local stabilizing and $\mathcal{F}$ is a linear function, we say that the system is locally stabilizing (for simplicity).

Given an $\mathcal{F}$-local stabilizing system and a system computation $\beta$ that starts at a state $q$ and reaches a legitimate state $q'$, the *perturbed set of nodes at* $q$, denoted as $PN(q)$, is defined as the maximal set of nodes that are in the same potentially perturbed set of nodes at $q$ and that change state from $q$ to $q'$. Formally, $PN(q) = \{i : i \in V.q \wedge q(i) \neq q'(i)\} \cap$ S', where S' $\in PP(q)$ and $|$S' $\cap \{i : i \in V.q \wedge q(i) \neq q'(i)\}| = max_{\mathrm{S} \in PP(q)}|\mathrm{S} \cap \{i : i \in V.q \wedge q(i) \neq q'(i)\}|$.

A node $i$ is *perturbed* at $q$ if $i \in PN(q)$, otherwise, it is *healthy* at $q$. A node is *contaminated* if it is healthy at $q$ and if the node executes at least one protocol action during stabilization. Then, the *range of contamination*, denoted by $R_c(q)$, is defined as the the maximum hop-distance from the set of contaminated nodes to the perturbed set of nodes $PN(q)$. That is,

$$R_c(q) \ = \ \max\nolimits_{i \in S_c} hops(i, PN(q))$$

where

$$\begin{aligned} S_c = \{i : i \in V.q \wedge i \text{ is healthy at } q \wedge \text{some protocol} \\ \text{action is executed at } i \text{ during stabilization}\}, \\ hops(i, PN(q)) = \min\nolimits_{j \in PN(q)} hops(i, j, G.q), \\ hops(i, j, G.q) = \text{the number of hops in the shortest} \\ \text{path between } i \text{ and } j \text{ in } G.q. \end{aligned}$$

By definition, the range of contamination $R_c(q)$ denotes the distance to which the perturbation at $q$ propagates during stabilization, thus $R_c(q)$ should be 0 ideally or be a function of the perturbation size at $q$ in practice.

## 6.3.2 Properties of $\mathcal{F}$-local stabilizing systems

A set of nodes $S$ are *contiguous* at a system state $q$ if $S \subseteq V.q$ and the subgraph of $G.q(V.q, E.q)$ on $S$ is connected, i.e., the graph $G'(V', E')$ is connected, where $V' = S$ and $E' = \{(i, j) : i \in S \wedge j \in S \wedge (i, j) \in E.q\}$. A maximal set of perturbed nodes that are contiguous is called a *perturbed region*. Then the following properties hold for a $\mathcal{F}$-local stabilizing system $G$:

- Starting at an arbitrary state $q$, the maximum distance that faults can propagate outward from the perturbed regions is $O(\mathcal{F}(P(q)))$, i.e., the range of contamination is $O(\mathcal{F}(P(q)))$. Therefore, every node that is $\omega(\mathcal{F}(P(q)))$ hops away from the perturbed regions at state $q$ will not be contaminated by the perturbation. (This claim comes from the observation that the time taken for a distributed algorithm to stabilize is at least proportional to the distance information propagates in the algorithm.)

- Starting at an arbitrary state $q$ where the perturbed regions are $\omega(\mathcal{F}(P(q)))$ hops away from one another, the stabilization of one perturbed region is independent of and concurrent with that of the other perturbed regions, and the time taken for the system to stabilize only depends on the size of the largest perturbed region.

- The availability of an $\mathcal{F}$-local stabilizing system is high in the sense that it stabilizes quickly after perturbations and the impact of perturbations is contained locally around where they occur.

## 6.4 Protocol LSRP

In this section, we first specify the problem of local stabilization in shortest path routing. Then we explain the limitations of existing distance-vector routing protocols, present the protocol concepts underlying LSRP, and finally present the design of LSRP.

### 6.4.1 Problem statement

The problem is to design a protocol that, given a system $G(V, E, W)$ and a destination node $r \in V$, constructs and maintains a spanning tree $T_G$ (called *shortest path tree*) of $G$ that meets the following requirements:

- Node $r$ is the root of the shortest path tree $T_G$;
- $(\forall i : i \in V \Rightarrow dist(i, r, T_G) = dist(i, r, G))$, where $dist(i, r, T_G)$ and $dist(i, r, G)$ are the minimum distance between nodes $i$ and $r$ in $T_G$ and $G$ respectively; (that is, the path from every node $i$ to $r$ in $T_G$ is a shortest path between $i$ and $r$ in $G$.)
- The system $G$ is $\mathcal{F}-$local stabilizing.

### 6.4.2 Fault propagation in existing distance-vector protocols

Existing distance-vector routing protocols are based on the distributed Bellman-Ford algorithm [46, 67]. In these protocols, each node $i$ maintains the distance, denoted as $d.i$, of and the next-hop, denoted as $p.i$, on its shortest path to each destination. For a destination $r$, if node $j$ is a neighbor of $i$ and $d.j = \min\{d.k : k \text{ is a neighbor of } i\}$, $i$ will choose $j$ as the next hop on its shortest path to $r$ (i.e., set $p.i$ to $j$) and set $d.i$ to $d.j + w.i.j$. However, in these protocols, faults cannot be contained around where they have occurred, and $\mathcal{F}$-local stabilization is not guaranteed, which results in routing instability.

One example is shown in Figure 6.2. For the same system in Figure 6.1, Figure 6.2(a) represents a system state where the state of node $v_8$ is corrupted such that $d.v_8 = 1$. Ideally, $v_8$ should correct its state such that $d.v_8 = 3$, and all the other nodes in the system remain unaffected by the state corruption at $v_8$. However, in existing distance-vector routing protocols, it is possible that nodes $v_6$ and $v_5$ detect the change of $d.v_8$ before $v_8$

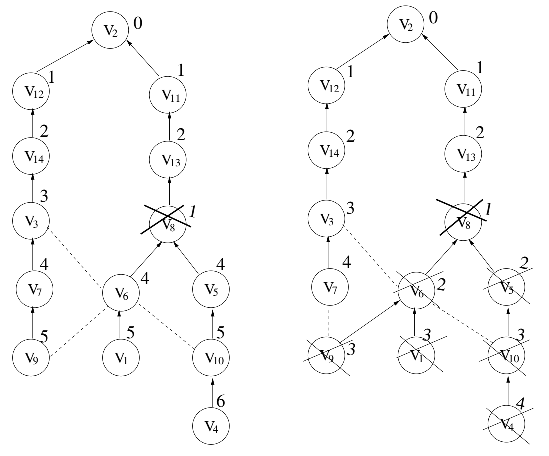


Figure 6.2: Example of fault propagation in existing distance-vector routing protocols

corrects its state. Then both $v_6$ and $v_5$ will change their state correspondingly such that $d.v_6 = d.v_5 = d.v_8 + 1 = 2$. And the same happens at nodes $v_9$, $v_1$, $v_{10}$, and $v_4$. Therefore, the fault at $v_8$ propagates to nodes $v_6$, $v_5$, etc., and the perturbed system state after the fault propagation from $v_8$ is shown in Figure 6.2(b). Even though the system will stabilize to a legitimate state later, nodes far away from $v_8$, such as $v_4$ and $v_9$, have been contaminated by the state corruption at $v_8$, and the time taken for the system to stabilize depends on the diameter of the system instead of the perturbation size. Furthermore, node $v_9$ has changed its route to destination $v_2$ because of the fault propagation, which leads to route flapping, a severe kind of routing instability.

### 6.4.3 Protocol concepts

In the example shown in Figure 6.2, the state corruption at $v_8$ can propagate far away until it reaches the leaves of the shortest path tree, and the time taken for the system to stabilize depends on its diameter instead of the perturbation size. The reasons for the unbounded fault propagation and slow stabilization are as follows:

- First, the distance value of $v_8$ (i.e., $d.v_8$) is corrupted to be smaller than it should be at any legitimate state;
- Second, before $v_8$ corrects its corrupted state, $v_6$ as well as $v_5$ detects that $d.v_8$ decreases. Because neither $v_6$ nor $v_5$ knows that the new state of $v_8$ is a corrupted one, both $v_6$ and $v_5$ update their state according to the corrupted state of $v_8$, and the state corruption at $v_8$ propagates to its neighbors $v_6$ and $v_5$. Then, the same thing that has happened to $v_6$ and $v_5$ happens to the neighboring nodes of $v_6$ and $v_5$, and so on.
- Third, after detecting that its state has been corrupted, $v_8$ corrects its state (i.e., sets $d.v_8$ to 3). Then, its neighbors $v_6$ and $v_5$ correct their corrupted states, and so on. However, this "correction" action is unable to catch up with the "fault propagation" action that propagates the initial corruption at $v_8$. Therefore, the initial corruption at $v_8$ is propagated far away until it reaches the leaves of the shortest path tree, hence the time taken for the system to stabilize depends on the system diameter instead of the perturbation size at the initial state.

In short, the reason why faults propagate and local stabilization is violated is that the "correction" action always lags behind the "fault propagation" action. Therefore, one approach to contain faults locally and achieve local stabilization is to guarantee that the node that is the source of fault propagation (for example, node $v_8$) will detect the existence of fault propagation, and initiate a "containment" action that can catch up with and stop the "fault propagation" action before faults propagate far away. We develop this approach as follows.

**Layering of diffusing waves.** In shortest path routing, the system computation is a diffusing computation by which nodes in the system learn their routes to a destination gradually. To achieve local stabilization, we design our protocol LSRP by layering the diffusing computation into three diffusing waves: the *stabilization wave*, the *containment wave*, and the *super-containment wave* (see Figure 6.3). Faults in a stabilization wave are contained by

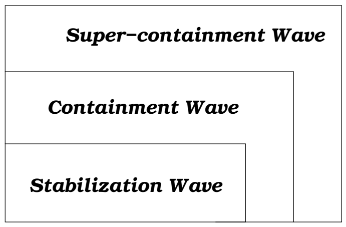


Figure 6.3: Layering of diffusing waves in shortest path routing

a containment wave, faults in a containment wave are contained by a super-containment wave, and each super-containment wave self-contains. To enable the above fault containment, containment waves propagate faster than stabilization waves, and super-containment waves propagate faster than containment waves.

The stabilization wave is a diffusing computation that implements the basic distributed Bellman-Ford algorithm with some changes to cooperate with the containment wave. A stabilization wave can propagate the "correction" action that enables a system to converge to a legitimate state, but a mistakenly initiated stabilization wave can propagate faults far away from where they initially occurred, as shown in Figure 6.2. To prevent a mistakenly initiated stabilization wave from propagating faults unboundedly, the containment wave is introduced.

**Containing the stabilization wave.** To enable fault containment and local stabilization, the source of fault propagation should detect the existence of the propagation and initiate a containment wave to stop it. In shortest path routing, each node chooses as its next-hop the neighbor that offers the least distance to the destination. Therefore, only "small distance value" propagates in distance-vector routing: if the distance value of a node is corrupted to be less than it could have been (via the distance value offered by the neighbors of the node), it is highly likely that a neighboring node will propagate the corrupted value by choosing the corrupted node as its next-hop; in contrast, if the distance value of a node is corrupted to be greater than it could have been, it is less likely that a neighboring node will propagate the corrupted value.

Therefore, we regard a node as a "potential source of fault propagation" (simply called source of fault propagation hereafter) if its distance value is less than what its neighboring nodes could have provided. Formally, a node $i$ is a source of fault propagation if, for every neighboring node $j$ of $i$ that is not involved in any containment wave, $d.j + w.i.j > d.i$ holds, and either $i$ is not the destination node, or $d.i$ is not equal to $0$. For example, node $v_8$ in Figure 6.2(a) is a source of fault propagation.

Whenever a node detects itself being a source of fault propagation, the node initiates a containment wave to stop the stabilization wave, if any, which has propagated the corrupted state of the node. The containment wave propagates along the same path as that by the stabilization wave. Since the containment wave propagates faster than the stabilization wave, it is able to catch up with and stop the stabilization wave.

Nevertheless, a containment wave can be mistakenly initiated due to state corruption. For example, in Figure 6.1, if the state of $v_{11}$ is corrupted such that $d.v_{11} = 2$, node $v_{13}$ will become a source of fault propagation and a containment wave will be initiated by $v_{13}$.

To prevent a mistakenly initiated containment wave from propagating unboundedly, the super-containment wave is introduced.

**Fault tolerance of the containment wave.** A node that has mistakenly initiated a containment wave will detect that it should not have initiated the containment wave after the perturbed regions have stabilized. Then it will initiate a super-containment wave that propagates along the same paths as those by the mistakenly initiated containment wave. Since the super-containment wave propagates faster than the containment wave, the super-containment wave will catch up with and stop the containment wave.

For the above wave-layering approach to work, the super-containment wave must self-stabilize itself locally upon perturbations; otherwise, there would be no end to the layering procedure. This is achieved by ensuring that the super-containment wave only uses variables defined for the stabilization wave and containment wave, and no extra variable is introduced for the super-containment wave.

**Loop freedom.** In the basic distributed Bellman-Ford algorithm, loops can form during stabilization, which leads to the bouncing effect and count-to-infinity problem [46] that delay the stabilization of a system and violate the time constraint of local stabilization. Therefore, in order to circumvent these two problems, our protocol avoids forming loops during stabilization, which, together with local fault containment, guarantees that the stabilization time is a function of the perturbation size in the worst case. Interestingly, loops can be avoided during stabilization just via the containment wave. The intuition is that a node that can select one of its descendants as its new parent (i.e. its next-hop) in the basic distributed Bellman-Ford algorithm becomes a source of fault propagation according to our definition. Therefore, a containment wave will be initiated at such a node, which guarantees loop freedom because no loop is formed in any containment wave.

### 6.4.4 The design of LSRP

The protocol LSRP (Locally Stabilizing shortest path Routing Protocol) is shown in Figure 6.4, where the constants, variables, and protocol actions for each node $i$ in a system are presented.

**Constants.** LSRP uses five constants: $r$, $d_s$, $d_c$, $d_{sc}$, and $d_{syn}$. $r$ is the ID of the destination node in a system to which all the other nodes in the system need to find the shortest path; $d_s$, $d_c$, and $d_{sc}$ are used to control the propagation speed of the stabilization wave, containment wave, and super-containment wave respectively; and $I_{syn}$ is used to control the frequency of information update between neighboring nodes.

**Variables.** As in existing distance-vector routing protocols, each node $i$ maintains the two variables $d.i$ and $p.i$, where $d.i$ records the distance from $i$ to $r$, and $p.i$ records the next-hop on the shortest path from $i$ to $r$ (i.e., the parent of $i$ in the shortest path tree rooted at $r$). (Note that, by definition, $d.i$ and $p.i$ are the only problem-specific variables for a node $i$ in LSRP.) To achieve local stabilization, each node $i$ also maintains a boolean variable $ghost.i$. $ghost.i$ is $true$ if node $i$ is being involved in a containment wave.

To enable inter-node coordination, $i$ maintains "mirror" variables $d.i'.i$, $p.i'.i$, and $ghost.i'.i$ for each neighbor $i'$, which denote $i$'s knowledge of the latest values of $d.i'$, $p.i'$, and $ghost.i'$ respectively. (For convenience in presentation, we also regard $d.i.i$ as $d.i$, $p.i.i$ as $p.i$, and $ghost.i.i$ as $ghost.i$.) To control the frequency of information update between neighboring nodes, variable $t.i$ is used to denote the time when $i$ broadcasts the values of $d.i$, $p.i$, and $ghost.i$ last time.

```
Protocol    $LSRP.i$
Constant    $r$ : node-id
            $d_s,\ d_c,\ d_{sc},\ I_{syn}$ : real
Var         $d.i,\ d.i'.i\ (i' \in N.i)$ : integer
            $p.i,\ p.i'.i\ (i' \in N.i)$ : node-id
            $ghost.i,\ ghost.i'.i\ (i' \in N.i)$ : boolean
            $t.i$ : real
            $k$ : node-id
Parameter   $j$ : node-id
Actions
  $\langle S_1 \rangle :: MP.i \wedge p.i \neq i \longrightarrow p.i := i;$
                                  $t.i := CLK.i;$ send $m(p.i)$ to $N.i$
  []
  $\langle S_2 \rangle :: SW.i.j \wedge \neg ghost.j.i \xrightarrow{d_s} d.i, p.i := d.j.i + w.i.j, j;$
                                           $ghost.i := false;$
                                           $t.i := CLK.i;$ send $m(d.i, p.i, ghost.i)$ to $N.i$
  []
  * * *
  $\langle C_1 \rangle :: \neg ghost.i \wedge (SP.i \vee CW.i) \xrightarrow{d_c} ghost.i := true;$
                                                 if $SP.i \rightarrow p.i := i$ fi;
                                                 $t.i := CLK.i;$ send $m(p.i, ghost.i)$ to $N.i$
  []
  $\langle C_2 \rangle :: ghost.i \wedge \neg(\exists k : k \in N.i \wedge p.k.i = i \wedge d.k.i = d.i + w.i.k) \longrightarrow$
                                         $ghost.i := false;$
                                         if $i = r \rightarrow d.i, p.i := 0, i$
                                            []
                                            $i \neq r \wedge PS.i.j \rightarrow d.i, p.i := d.j.i + w.i.j, j$
                                            []
                                            $i \neq r \wedge \neg(\exists k : PS.i.k) \rightarrow d.i, p.i := \infty, i$
                                         fi;
                                         $t.i := CLK.i;$ send $m(d.i, p.i, ghost.i)$ to $N.i$
  []
  * * *
  $\langle SC \rangle :: ghost.i \wedge SCW.i \xrightarrow{d_{sc}} ghost.i := false;$
                                         if $p.i = i \rightarrow$
                                            do $d.k.i + w.i.k = d.i \wedge SW.i.k \rightarrow p.i := k$ od;
                                         fi;
                                         $t.i := CLK.i;$ send $m(ghost.i)$ to $N.i$
  []
  * * *
  $\langle SYN_1 \rangle :: (t.i + I_{syn} \leq CLK.i) \vee (t.i > CLK.i) \longrightarrow t.i := CLK.i;$ send $m(d.i, p.i, ghost.i)$ to $N.i$
  []
  $\langle SYN_2 \rangle ::$ rcv $m$ from $j \longrightarrow$ update $d.j.i$, $p.j.i$, and/or $ghost.j.i$ accordingly
```

Figure 6.4: LSRP: local stabilization in shortest path routing

For clarity of presentation, we let $N.i$ be the set of neighboring nodes of $i$, we let $CLK.i$ be the clock value of $i$, and we let $w.i.i$ be $\infty$ (i.e., there is no self-loop). A dummy variable $k$ is also used.

**Protocol actions.** As discussed in Section 6.4.3, the diffusing computation in LSRP consists of three diffusing waves: the stabilization wave, the containment wave, and the super-containment wave. These diffusing waves are implemented in LSRP as follows:

- Actions $S_1$ and $S_2$ implement the stabilization wave. More specifically, $S_2$ implements the distributed Bellman-Ford algorithm; $S_1$ guarantees that the next-hop of a node $i$ (i.e., $p.i$) is consistent with the information within its neighborhood (i.e., $S_1$ is introduced to guarantee self-stabilization).

- Actions $C_1$ and $C_2$ implement the containment wave. To enable a super-containment wave to trace a mistakenly initiated containment wave by the parent-child relationship between neighboring nodes, each containment wave is a *round* procedure consisting of a phase of propagating outward and a phase of shrinking back. Action $C_1$ implements the phase of propagating outward which is to stop the corresponding stabilization wave, and action $C_2$ implements the phase of shrinking back after the stabilization wave has been stopped.

- Action $SC$ implements the super-containment wave. The super-containment wave propagates along the path signified by the parent-child relationship maintained in the corresponding containment wave.

The propagation speed of a diffusing wave is controlled by the guard hold-time of the actions implementing the wave. To guarantee that containment waves propagate faster than stabilization waves and that super-containment waves propagate faster than containment waves in the presence of clock drift as well as message passing delay, the guard hold-time $d_s$, $d_c$, and $d_{sc}$ used in LSRP should be such that $d_s > \alpha \cdot (d_c + U)$, $d_c > \alpha \cdot (d_{sc} + U)$, and $d_{sc} \geq 0$. (For clarity of presentation, we relegate the detailed reasoning to [16].)

To update information between neighboring nodes, actions *SYN*$_1$ and *SYN*$_2$ are used.

We further elaborate on the protocol actions as follows.

***Stabilization wave.*** By implementing the distributed Bellman-Ford algorithm with some changes to cooperate with containment wave, the stabilization wave guarantees that a system eventually stabilizes to a legitimate state.

$\underline{Action\ S_1}$: if node $i$ is a minimal point (i.e., $MP.i = true$) but $p.i \neq i$, it sets $p.i$ to $i$. Then, $i$ sets $t.i$ to its current clock value, and broadcasts the new value of $p.i$ to its neighbors.

$MP.i$ is defined as

$$(i = r \wedge d.i = 0) \vee (ghost.i \wedge SP.i), \text{ where } SP.i = true \text{ if } i \text{ is a source of fault propagation.}$$

That is, a node $i$ is a minimal point if it is the destination node and $d.i = 0$, or if it has initiated a containment wave that has not finished.

$\underline{Action\ S_2}$: if node $i$ should propagate a stabilization wave from node $j$ (i.e., $SW.i.j = true$) that is not being involved in any containment wave, and this condition continuously held in the past $d_s$ time, then $i$ sets $j$ as its parent, and sets $d.i$, $ghost.i$ to $d.j.i + w.i.j$, $false$ respectively. Also, $i$ sets $t.i$ to its current clock value, and broadcasts the new values of $d.i$, $p.i$, and $ghost.i$ to its neighbors.

$SW.i.j$ is defined as

$$\begin{array}{l} j \in N.i \ \wedge \ d.j.i + w.i.j \leq d.i \ \wedge \\ (\forall k : k \in N.i \Rightarrow d.j.i + w.i.j \leq d.k.i + w.i.k) \ \wedge \\ ((j \neq p.i \wedge (p.i \in N.i \wedge \neg ghost.(p.i).i \Rightarrow \\ \qquad\qquad d.j.i + w.i.j < d.(p.i).i + w.i.(p.i))) \\ \ \vee \\ \ (j = p.i \wedge d.i \neq d.j.i + w.i.j)) \end{array}$$

That is, node $i$ should propagate a stabilization wave from node $j$ if

- ◦ $j$ is the neighbor of $i$ via which the distance value of $i$ is the smallest; and
- ◦ if $j$ is not the current parent of $i$, then the distance value of $i$ via $j$ is less than that via $p.i$ unless $p.i$ is not a neighbor of $i$ or is involved in a containment wave; if $j$ is the current parent of $i$, then the distance value of $i$ is not equal to that of $j$ plus $w.i.j$.

However, node $i$ should not propagate any stabilization wave from $j$ if $j$ is being involved in a containment wave, since the state of any node being involved in a containment wave is corrupted.

***Containment wave.*** The containment wave prevents a stabilization wave from propagating faults far away from where they have occurred.

$\underline{\mathit{Action}\ C_1}$: if node $i$ is not being involved in any containment wave (i.e., $ghost.i = false$), but it is either a source of fault propagation (i.e., $SP.i = true$) or it should propagate a containment wave from its parent (i.e., $CW.i = true$), and this condition continuously held in the past $d_c$ time, then $i$ sets $ghost.i$ to $true$ in order to initiate or propagate a containment wave. Moreover, if $i$ is a source of fault propagation, it sets $p.i$ to $i$.[22] Then, $i$ sets $t.i$ to its current clock value, and broadcasts the new values of $p.i$ and $ghost.i$ to its neighbors.

$SP.i$ is defined as

$$\begin{array}{l}(\forall j : j \in N.i \wedge \neg ghost.j.i \Rightarrow d.j.i + w.i.j > d.i) \wedge \\ ((i \neq r \wedge d.i \neq \infty \wedge d.i \neq d.(p.i).i + w.i.(p.i)) \vee \\ \ (i = r \wedge d.i \neq 0))\end{array}$$

That is, a node $i$ is a source of fault propagation if none of its neighbors that are not involved in any containment wave can offer $i$ a distance value that is no greater than what $i$ currently

[22]Conceptually, when $i$ is a source of fault propagation, it is a local minimum in terms of distance values and no neighbor can act as its next-hop by providing a smaller distance to the destination. Therefore, $i$ sets $p.i$ to $i$. By this design, the destination node $r$ can "stabilize" $p.r$ to $r$ when $d.r \neq 0$; a node $i$ that is in a loop (e.g., due to state corruption) can break the loop within constant time by setting $p.i$ to itself.

has, and either $i$ is not the destination node and its distance value is not consistent with that of its parent, or $i$ is the destination node and its distance value is not $0$.

$CW.i$ is defined as

$$\begin{array}{l} p.i \in N.i \wedge ghost.(p.i).i \wedge d.i = d.(p.i).i + w.i.(p.i) \wedge \\ \neg(\exists k : k \in N.i \wedge \neg ghost.k.i \wedge d.k.i + w.i.k \leq d.i) \end{array}$$

That is, a node $i$ should propagate a containment wave from its parent $p.i$ if $p.i$ is a neighbor of $i$ and is involved in a containment wave, $i$ has copied the corrupted distance value of $p.i$ (i.e., $d.i = d.(p.i).i + w.i.(p.i)$), and $i$ does not have a neighbor $k$ that is not involved in any containment wave and can offer $i$ a distance value smaller than what $i$ currently has.

$\underline{Action\ C_2}$: if node $i$ is involved in a containment wave, but $i$ has no child $k$ that is perturbed due to the state corruption at $i$ (i.e., $p.k.i = i$ and $d.k.i = d.i + w.i.k$), then $i$ sets $ghost.i$ to $false$, and

- ∘ if $i$ is the destination node, then $i$ sets $d.i$ and $p.i$ to $0$ and $i$ respectively;
- ∘ if $i$ is not the destination node, then $i$ sets $d.i$ and $p.i$ to $d.j.i+w.i.j$ and $j$ respectively, if there exists a parent substitute $j$ of $i$ (i.e., $PS.i.j = true$); otherwise, $i$ sets $d.i$ and $p.i$ to $\infty$ and $i$ respectively to guarantee loop freedom during stabilization.

  $PS.i.j$ is defined as

$$\begin{array}{l} j \in N.i \wedge \neg ghost.j.i \wedge p.j.i \neq i \wedge \\ d.j.i + w.i.j \leq d.i \wedge \\ (\forall k : k \in N.i \wedge \neg ghost.k.i \Rightarrow \\ \qquad\qquad\qquad d.j.i + w.i.j \leq d.k.i + w.i.k) \end{array}$$

  That is, node $j$ is a parent substitute of $i$ if $j$ is not a child of $i$, and, among all the neighbors of $i$ that are not involved in any containment wave, $j$ offers $i$ the smallest distance value that is no greater than what $i$ currently has.

Also, $i$ sets $t.i$ to its current clock value, and broadcasts the new values of $d.i$, $p.i$, and $ghost.i$ to its neighbors.

Action $C_2$ guarantees that a containment wave will shrink back to its initiator after the containment wave has caught up with and stopped the stabilization wave that propagates the faults which initially occurred at the initiator of the containment wave.

***Super-containment wave.*** The super-containment wave prevents a mistakenly initiated containment wave from propagating unbounded, and corrects faults in the containment wave.

$\underline{Action\ SC}$: if node $i$ is involved in a containment wave (i.e., $ghost.i = true$), but it should initiate or propagate a super-containment wave from its parent (i.e., $SCW.i = true$), and this condition continuously held in the past $d_{sc}$ time, then $i$ sets $ghost.i$ to $false$. Moreover, if $i$ has set $p.i$ to itself because it was a source of fault propagation, $i$ recovers its parent immediately. Then, $i$ sets $t.i$ to its current clock value, and broadcasts the new value of $ghost.i$ to its neighbors.

$SCW.i$ is defined as

$$\begin{array}{l}(i = r \wedge d.i = 0) \vee \\ (i \neq r \wedge \neg SP.i \wedge (p.i \neq i \Rightarrow \neg ghost.(p.i).i))\end{array}$$

That is, $i$ should initiate or propagate a super-containment wave if

- it is the destination node and $d.i = 0$; or
- it is not the destination node, and neither is it a source of fault propagation nor is its parent involved in any containment wave.

***Information update.*** Actions *SYN*$_1$ and *SYN*$_2$ guarantee that the values of mirror variables $d.i'.i$, $p.i'.i$, and $ghost.i'.i$ at node $i$ stabilize to those of $d.i'$, $p.i'$, and $ghost.i'$ for every neighbor $i'$ of $i$.

*Action SYN*$_1$: if $i$ did not broadcast the values of $d.i$, $p.i$, and $ghost.i$ in the past $I_{syn}$ time (i.e., $t.i + I_{syn} \leq CLK.i$) or if variable $t.i$ is corrupted to be greater than the current clock value of $i$, $i$ sets $t.i$ to its current clock value, and broadcasts the values of $d.i$, $p.i$, and $ghost.i$ to its neighbors.

*Action SYN*$_2$: when $i$ receives the latest values of $d.j$, $p.j$, and/or $ghost.j$ from its neighbor $j$, $i$ records the values to variables $d.j.i$, $p.j.i$, and/or $ghost.j.i$ respectively.

### 6.4.5 Examples revisited

To illustrate how LSRP behaves in the presence of faults, we reconsider the examples discussed in previous sections. For simplicity of presentation, we assume that $\alpha = 1$, link delay is a constant $u$, processing delay is negligible, containment waves propagate twice as fast as stabilization waves (i.e, $d_s = 2d_c + u$), and super-containment waves propagate four times as fast as containment waves (i.e., $d_c = 4d_{sc} + 3u$).

We first study the case where $d.v_8$ is corrupted to 1 and nodes $v_6$ and $v_5$ have learned the corrupted value when the system is at the state as shown in Figure 6.5. The system behavior after the state corruption at $v_8$ is shown by the space-time diagram in Figure 6.5:

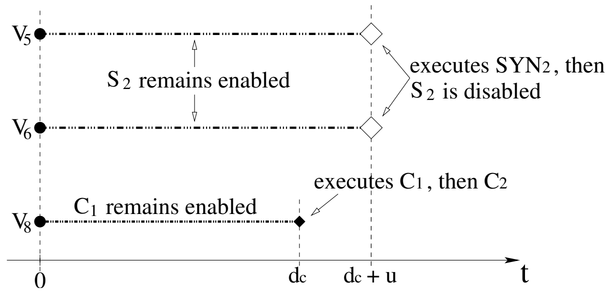


Figure 6.5: System behavior after $d.v_8$ is corrupted to 1.

- First, the guard for action $C_1$ evaluates to true at $v_8$ (because $v_8$ is a source of fault propagation, by definition), and the guard for action $S_2$ evaluates to true at $v_6$ and $v_5$. Therefore, $C_1$ becomes enabled at $v_8$ and $S_2$ becomes enabled at $v_6$ and $v_5$. For convenience, we regard this moment as time $0$.

- $C_1$ remains enabled at $v_8$ until time $d_c$, when $v_8$ executes $C_1$. After the execution of $C_1$, $C_2$ becomes enabled at $v_8$ and is executed immediately, since the guard hold-time for $C_2$ is zero. The execution of $C_2$ corrects $d.v_8$ to 3.

- $C_2$ remains enabled at $v_6$ and $v_5$ until time $d_c + u$, when $v_6$ and $v_5$ receive messages from $v_8$ reflecting its new state. Therefore, action *SYN*$_2$ is executed twice (processing the two messages from $v_8$) at $v_6$ and $v_5$ at time $d_c + u$, which disables $S_2$ at $v_6$ and $v_5$. At this moment, the system reaches a legitimate state.

In the above process, only actions $C_1$ and $C_2$ are executed at $v_8$, and no action is executed elsewhere. Therefore, no other nodes in the system is affected by the state corruption at $v_8$, which is the ideal result achievable.

Faults may not be ideally contained in all cases, but they are always contained locally around where they occur in LSRP. To illustrate this, we consider the case where $d.v_{11}$ is corrupted to 2 and $v_{13}$ has learned the corrupted value when the system is at the state as shown in Figure 6.1. The system behavior after the state corruption at $v_{11}$ is shown in Figure 6.6:

- First, action $S_2$ becomes enabled at $v_{11}$, and action $C_1$ becomes enabled at $v_{13}$. For convenience, we regard this moment as time 0.

- At time $d_c$, $v_{13}$ executes $C_1$ and sends its new state to $v_8$ (by which the containment wave propagates from $v_{13}$ to $v_8$).

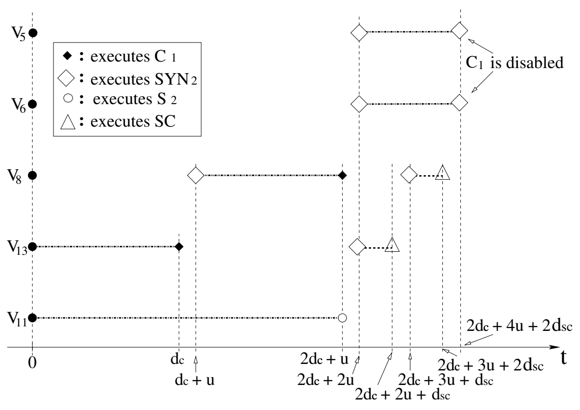


Figure 6.6: System behavior after $d.v_{11}$ is corrupted to 2.

- The new state of $v_{13}$ reaches $v_8$ at time $d_c + u$, when action *SYN*$_2$ is enabled and executed immediately at $v_8$. As a result, action $C_1$ becomes enabled at $v_8$ at time $d_c + u$.

- Then, $v_8$ executes $C_1$ and $v_{11}$ executes $S_2$ at time $2d_c + u$. $u$ time later, $v_6$ and $v_5$ receive the new state of $v_8$, and $v_{13}$ learns the new state of $v_{11}$. Consequently, $C_1$ becomes enabled at $v_6$ and $v_5$, and $SC$ becomes enabled at $v_{13}$.

- At time $2d_c + 2u + d_{sc}$, $v_{13}$ executes $SC$ and sends its corrected state to $v_8$ (by which the super-containment wave propagates from $v_{13}$ to $v_8$). As a result, $SC$ becomes enabled at $v_8$ at time $2d_c + 2u + d_{sc}$.

- $d_{sc}$ time later, $v_8$ executes $SC$ and sends its new state to $v_6$ and $v_5$ (by which the super-containment wave propagates to $v_6$ and $v_5$).

- At time $2d_c + 4u + 2d_{sc}$, action *SYN*$_2$ becomes enabled and is executed immediately at $v_6$ and $v_5$, which in turn disables $C_1$ at $v_6$ and $v_5$. As a result, the system reaches its legitimate state.

In the above process, only nodes $v_{11}, v_{13}$, and $v_8$ execute one or more protocol actions, while the rest of the system remain unaffected. Therefore, the impact of the state corruption at $v_{11}$ is contained locally (i.e., within 2 hops in the above case).

## 6.5 Protocol analysis

In this section, we present the property of local stabilization in a system where LSRP is used. We also present the properties of loop freedom during stabilization and quick loop removal in LSRP, which are not only necessary for local stabilization in routing, but also critical for improving network resource utilization and quality of communication. (For clarity of presentation, we relegate the proofs of all the theorems and lemmas in this chapter to the appendix D and [16]).

### 6.5.1 Property of local stabilization

Given a system topology $G'(V', E')$, we define predicate $\mathcal{L}$ as

$$\begin{array}{l}(\forall i : i \in V' \Rightarrow \neg ghost.i \wedge LH.i) \wedge \\ (\forall e : e \in E' \Rightarrow \textit{there is no message in } e)\end{array}$$

where $LH.i$ is defined as

$$\begin{array}{l}(i = r \Rightarrow d.i = 0 \wedge p.i = i) \wedge \\ (i \neq r \Rightarrow d.i = d.(p.i) + w.i.(p.i) \wedge p.i \in N.i \wedge \\ \qquad\qquad (\forall k : k \in N.i \Rightarrow d.(p.i) + w.i.(p.i) \leq d.k + w.i.k))\end{array}$$

Then, every state in $\mathcal{L}$ is a legitimate system state where the shortest path tree rooted at the destination node $r$ is formed (by variable $p.i$ at every node $i$ in the system), and every node $i$ has learned the distance and the next-hop on its shortest path to $r$. For LSRP, we have

**Theorem 2 (Self-stabilization)** *Starting at an arbitrary state, every computation of a system where LSRP is used is guaranteed to reach a state in $\mathcal{L}$.* $\square$

From Theorems 2, we know that LSRP guarantees the formation of the shortest path tree in a system when it starts at an arbitrary state.

Furthermore, local stabilization is guaranteed in LSRP. The analysis is as follows.

When there is only one perturbed region at the initial state, we have

**Lemma 1** *Starting at an arbitrary state $q_0$ where there is only one perturbed region, every system computation reaches a state in $\mathcal{L}$ within $O(P(q_0))$ time, and the range of contamination is $O(P(q_0))$.* $\square$

When the set of perturbed nodes are not contiguous, there are multiple perturbed regions (denoted as $S_0$, $S_1$, ..., $S_m$, $m \geq 1$) in the system. For each perturbed region $S_i$, we define its *containment region* $CR_i$ as the union of $S_i$ and the set of nodes that are contaminated during stabilization because of the existence of $S_i$. Two containment regions $CR_i$ and $CR_j$ are *disjoint* if there do not exist any two neighboring nodes $k$ and $k'$ such that $k \in CR_i$ and $k' \in CR_j$, otherwise, they are *adjoining*. Multiple containment regions $CR_0, CR_1, \ldots, CR_m$ ($m \geq 1$) are disjoint if there do not exist any two containment regions $CR_i$ and $CR_j$ ($i \neq j$) that are adjoining. Then, we have

**Lemma 2** *Starting at an arbitrary state $q_0$ where the perturbed regions are $S_0, S_1, \ldots, S_m$ and their containment regions are disjoint, every system computation reaches a state in $\mathcal{L}$ within $O(\max_{i \in 0..m} |S_i|)$ time, and the range of contamination is $O(\max_{i \in 0..m} |S_i|)$.* $\square$

Given any two perturbed regions $S_i$ and $S_j$ ($i \neq j$) at a state $q$, the half-distance between $S_i$ and $S_j$ is half of the minimum distance from a node in $S_i$ to another node in $S_j$, that is, $\min_{k \in S_i, k' \in S_j} \lfloor \frac{dist(k,k',G.q)}{2} \rfloor$, where $dist(k, k', G.q)$ denotes the minimum distance between node $k$ and $k'$ in graph $G.q$. Then, Lemma 2 implies

**Corollary 1** *Starting at an arbitrary state $q_0$ where the perturbed regions are $S_0, S_1, \ldots, S_m$ and the half-distance between any two of them is $\omega(\max_{i \in 0..m} |S_i|)$, every system computation reaches a state in $\mathcal{L}$ within $O(\max_{i \in 0..m} |S_i|)$ time, and the range of contamination is $O(\max_{i \in 0..m} |S_i|)$.* $\square$

Multiple containment regions $CR_0$, $CR_1$, ..., $CR_m$ ($m \geq 1$) are adjoining if, for any two containment regions $CR_i$ and $CR_j$ ($i \neq j$), either $CR_i$ and $CR_j$ are adjoining or there exist a sequence of containment region $CR_{k_0}, CR_{k_1}, \ldots, CR_{k_t}$ such that $CR_i$ are adjoining with $CR_{k_0}$, $CR_{k_n}$ are adjoining with $CR_{k_{n+1}}$ ($n = 0, \ldots, t-1$), and $CR_{k_t}$ are adjoining with $CR_j$. Then, we have

**Lemma 3** *Starting at an arbitrary state $q_0$ where the perturbed regions are $S_0, S_1, \ldots, S_m$ and their containment regions are adjoining, every system computation reaches a state in $\mathcal{L}$ within $O(\sum_{i=0}^{m} |S_i|)$ time, but the range of contamination is still $O(\max_{i \in 0..m} |S_i|)$.* $\square$

Lemmas 2 and 3 imply

**Corollary 2** *Starting at an arbitrary state $q_0$ where the perturbed regions are $S_0, S_1, \ldots, S_m$, every system computation reaches a state in $\mathcal{L}$ within $O(P(q_0))$ time, and the range of contamination is $O(\max_{i \in 0..m} |S_i|)$ (which is $o(P(q_0))$).* $\square$

Lemma 1 and Corollary 2 imply

**Theorem 3 (Local stabilization)** *Starting at an arbitrary state $q_0$, every system computation reaches a state in $\mathcal{L}$ within $O(P(q_0))$ time, and the range of contamination is $O(\mathrm{MAXP})$, where $\mathrm{MAXP}$ denotes the number of nodes in the largest perturbed region at $q_0$ and is $o(P(q_0))$. That is, the system is $\mathcal{F}$-local stabilizing, where $\mathcal{F}$ is a linear function.* □

By Theorem 3, we see that LSRP solves the shortest path routing problem in a linear-local stabilizing manner.

### 6.5.2 Properties of loop freedom and quick loop removal

**Theorem 4 (Loop freedom)** *Starting at an arbitrary state where there is no loop, every system computation reaches a state in $\mathcal{L}$, and there is no loop at any state along the computation.* □

From Theorem 4, we see that there is no loop in the system during stabilization if the only possible fault in a system is node fail-stop, because no loop can be formed just by node fail-stop, and there is no loop at any initial state of a system computation if the only fault is node fail-stop.

**Theorem 5 (1-round loop breakage)** *Starting at an arbitrary state where there exists at least one loop, every system computation reaches a state where there is no loop after at most one round of computation.* □

Theorems 4 and 5 imply

**Corollary 3** *Starting at an arbitrary state where there exists at least one loop, every system computation reaches a state where there is no loop after at most $(d_s + U)$ time.* □

## 6.6 Discussion

In this section, we discuss the impact of network topology on local stabilization, and we discuss issues related to the application of LSRP.

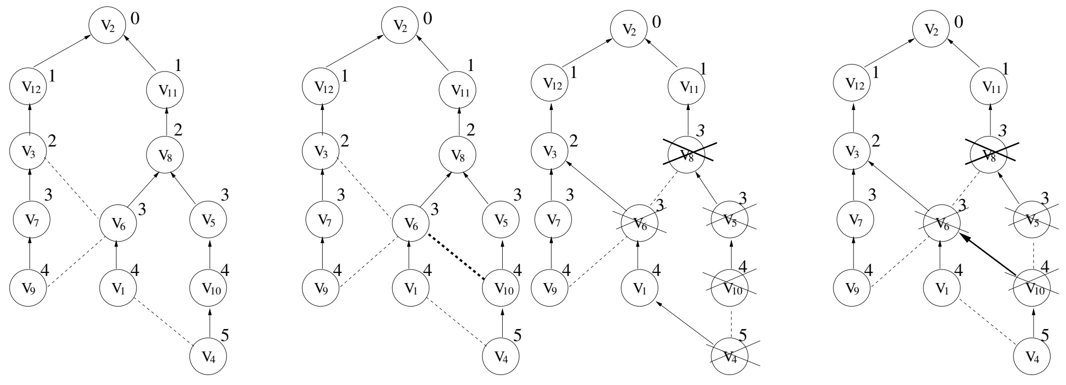
### 6.6.1 Impact of network topology on local stabilization

For the problem of shortest path routing, the network topology of a system can affect the perturbation size, the range of contamination, and the self-stabilization time in the sense that higher edge density is conducive to local stabilization.

Given a system topology $G.q_0(V.q_0, E.q_0)$ at state $q_0$, if we add some edges to $G.q_0$ and obtain another system topology $G.q'_0(V.q'_0, E.q'_0)$ at state $q'_0$ with denser edges (i.e. $E.q_0 \subset E.q'_0$), then for every node that is both in $G.q_0$ and in $G.q'_0$, the number of different shortest paths to the destination node in $G.q_0$ will be no more than that in $G.q'_0$. Then, if the same node $i$ fail-stops in both $G.q_0$ and $G.q'_0$, and $G.q_0$, $G.q'_0$ transit to $G.q$, $G.q'$ respectively, the number of nodes that are perturbed due to the fail-stop of $i$ in $G.q$ will be no less than that in $G.q'$. More generally, if the same faults occur when the system is at $q_0$ and at $q'_0$, and the system reaches $q$ and $q'$ respectively after the faults, the perturbation size at state $q$ will be no less than that at $q'$. Moreover, even if the perturbation sizes at $q$ and $q'$ are the same, a mistakenly initiated containment wave, if any, will propagate no farther in $G.q'$ than in $G.q$ Therefore, the time taken for the system to stabilize from $q$ and the range of contamination during stabilization are no less than that with respect to $q'$. Formally,

**Proposition 1** *Given a system $G$ and two system states $q$ and $q'$ such that $V.q = V.q'$, $E.q \subseteq E.q'$, and $(\forall i : i \in V.q \Rightarrow q(i) = q'(i))$, then $P(q) \geq P(q')$, $R_c(q) \geq R_c(q')$, and the time taken for $G$ to stabilize from $q$ is no less than that with respect to $q'$.* $\square$

One example is shown in Figure 6.7. Figure 6.7(a) and (b) represent system topology $G.q_0$



(a) System topology $G.q_0$

(b) System topology $G.q_0'$

(c) Perturbed state in $G.q_0$

(d) Perturbed state in $G.q_0'$

Figure 6.7: An example where denser edges reduce the perturbation size and range of contamination upon perturbations in the system

and $G.q_0'$ at legitimate states $q_0$ and $q_0'$ respectively. The only difference between $G.q_0$ and $G.q_0'$ is that edge $(v_6,\ v_{10})$ is not in $G.q_0$ but is in $G.q_0'$. Then,

- If node $v_8$ fail-stops at both state $q_0$ and $q_0'$, and the system reaches state $q$ and $q'$ respectively, then the perturbation size at state $q$ is 4 (i.e., nodes $v_6$, $v_5$, $v_4$, and $v_{10}$ have to change their next-hops), whereas the perturbation size at $q'$ is 3 (i.e., nodes $v_6$, $v_5$, and $v_{10}$ have to change their next-hops, but $v_4$ does not);

- If the state of $v_8$ is corrupted in both $G.q_0$ and $G.q_0'$ such that $d.v_8 = 3$, then the perturbed states in $G.q_0$ and $G.q_0'$ during stabilization can be those as shown in Figure 6.7(c) and (d) respectively, by which we can see that the range of contamination in $G.q_0$ can be 3, whereas the range of contamination in $G.q_0'$ can only be 2 at most. The reason for this is that action $S_2$ is enabled at $v_{10}$ in $G.q_0'$ and the mistakenly initiated containment wave from $v_5$ will not propagate beyond $v_{10}$. Similarly, the time taken to stabilize in $G.q_0'$ is no more than that in $G.q_0$.

In wireless networks, especially in wireless sensor networks [99], the edges tend to be dense because of dense node distribution and wireless transmission property (i.e., nodes within transmission range of one another are connected with one another). Our conclusion, therefore, is that wireless (sensor) networks with higher edge density are likely to contain faults more tightly and to stabilize faster.

### 6.6.2 Issues related to the application of LSRP

**Accommodating continuous faults.** In a distributed system, faults may keep occurring in certain regions of the system with some frequencies or for certain periods of time. For example, congestion-induced route flapping may keep occurring at network edge routers for a certain period of time in Internet inter-domain routing [92]. Due to its property of local stabilization, LSRP also accommodates these kinds of faults, to an extent depending on the perturbation size, while still guaranteeing local fault containment and local stabilization. We elaborate on this as follows.

Theorem 3 implies

**Corollary 4** *If a transient fault that drives a system $G$ to state $q_0$ keeps occurring, but the interval between two consecutive occurrences of the fault is $\omega(P(q_0))$, then the range of contamination is $O(\mathrm{MAXP})$, where $\mathrm{MAXP}$ denotes the number of nodes in the largest perturbed region at $q_0$ and is $o(P(q_0))$.* □

From Corollary 4, we see that when a fault keeps occurring, the impact of the fault is still locally contained as long as the interval between two consecutive occurrences is asymptotically greater than the perturbation size of the fault.

Moreover, we have

**Theorem 6** *Starting at an arbitrary state $q_0$ at time $t_0$, if only transient faults occur, they occur only inside containment regions, and no fault occurs after time $t_0 + T$ ($T \geq 0$), then every system computation reaches a state in $\mathcal{L}$ within $O(P(q_0){+}T)$ time, and the range of contamination is $O(P(q_0) + T)$.* □

From Theorem 6, we see that, starting at $q_0$, the time taken for a system to stabilize and the range of contamination are still $O(P(q_0))$, if faults stop occurring in the containment regions $O(P(q_0))$ time after the system computation starts and no faults occur to nodes outside the containment regions.

**Speed of diffusing waves.** In LSRP, parameters $d_s$, $d_c$, and $d_{sc}$ control the propagation speed of stabilization waves, containment waves, and super-containment waves respectively. Therefore, the relationship between these three parameters determines the degree of fault containment in the presence of faults. Especially, we need to choose $d_s$ and $d_c$ carefully: on one hand, the larger the ratio $\frac{d_s}{d_c}$ is, the more tightly a mistakenly initiated stabilization wave is contained; on the other hand, the smaller the ratio $\frac{d_s}{d_c}$ is, the more tightly a mistakenly initiated containment wave tends to be contained (since the corresponding super-containment wave will not be initiated until certain stabilization wave is executed, as shown in Figure 6.6). In practice, we should consider the probabilities of stabilization or containment waves being mistakenly initiated and the degree of fault containment we expect in choosing the parameters.

**Control overhead.** The impact of faults is locally contained in LSRP, therefore the control overhead (e.g., the number of control messages) is also a function of the perturbation size, instead of the system size. Consequently, LSRP asymptotically reduces the control overhead when compared with non-locally-stabilizing protocols such as DUAL and LPA.

Moreover, the diffusing computation involved in stabilization and containment waves of LSRP also (implicitly) exists in other routing protocols (such as DUAL and LPA) to guarantee convergence as well as loop freedom. Therefore, stabilization and containment waves in LSRP do not introduce much more overhead except for the bit used to encode the variable $ghost$. Of course, the overhead associated with super-containment waves exists only in LSRP and not in other protocols; but this overhead is low because it only needs one bit to encode the variable $ghost$, and it is local in the sense that it is bounded from above by a function of the perturbation size.

## 6.7 Summary

To formally characterize properties of local stabilization in networked and distributed systems, we formulated the concepts of perturbation size, $\mathcal{F}$-local stabilization, and range of contamination. These concepts are generically applicable to networked and distributed systems, and are thus interesting in their own right.

For the problem of local stabilization in shortest path routing, we designed LSRP. LSRP guarantees both local stabilization and loop freedom during stabilization. In LSRP, we introduced delays in action execution to control the propagation speeds of diffusing waves. This does not slow down the convergence of a system, because the stabilization time is only a linear function of the perturbation size instead of the system size, which is especially desirable in large-scale systems where faults generally occur only at a small part of the system. Moreover, the method of introducing delays in action execution is also commonly used in Internet routing in order to reduce control overhead and routing flaps. For example, timer MinRouteAdvertisementInterval is used in BGP to control the frequency of route exchange between BGP peers. The timer is similar to the delay introduced for the stabilization wave in LSRP. In implementing LSRP, we only need to introduce smaller timers for the containment wave and super-containment wave.

We observed that higher edge density in a system can reduce the perturbation size, the range of contamination, and the self-stabilization time. This leads to the interesting question of how to design or self-configure a network such that the perturbation size, the range of contamination, and the self-stabilization time are minimized.

In the literature of network routing protocol design [46], formation of routing loops is regarded as problematic, and a variety of schemes have been proposed to avoid forming loops, such as those used in EIGRP, OSPF, and BGP. However, fault propagation and routing instability remain as problems in OSPF and BGP. The root cause appears to be that these protocols are not designed to tolerate such faults as misconfiguration and persistent congestion, which are special cases of state corruption. In LSRP, state corruption is dealt with by way of local stabilization. As a result, looping is implicitly avoided by taking loop-formation as a kind of state corruption, without introducing special mechanisms to deal with potential loops. By local stabilization, LSRP prevents faults from propagating far away and increases the stability as well as availability of a system. Therefore, the question of whether we should take various kinds of faults as state corruption and deal with them by way of (local) stabilization deserves further exploration.

# CHAPTER 7

# RELATED WORK

In this chapter, we discuss the literature related to the topics considered in this dissertation, that is, messaging architecture, wireless link estimation and routing, packing-oriented scheduling, transport control, and local stabilization.

## 7.1 Messaging architecture

Sensornet protocol SP [77] provides a unified link layer abstraction for sensornets. Focusing on the interface between link and network layers, SP is complementary to our focus on higher layer architecture issues, and SP can be incorporated into the SMA architecture. Woo et al. [96] discussed networking support for query processing in sensor networks. Issues such as query-oriented routing, efficient rendezvous for storage and correlation, and unified in-network system have been discussed. While focusing on query processing, [96] does not concentrate on the architectural and algorithmic issues to support a broader range of applications such as distributed signal processing and computing. To adapt the communication protocol to the changing network conditions and application requirements, Impala [65] used the Adaptation Finite State Machine (AFSM) to control the adaptation of communication protocols. To provide application-specific QoS in ubiquitous environments, Nahrstedt et al. [71] proposed a framework for QoS specification and compilation, QoS setup, and QoS adaptation. Our work complements those in [65] and [71] by focusing on issues such as application-adaptive link estimation, structuring, and scheduling which are autonomous without human in the loop.

Mechanisms have been proposed in [25] and [64] for directing data queries to where information is via information-directed routing. Our work complements [64] by considering the architectural issues in application-adaptive messaging as well as the algorithmic issues in application-adaptive structuring and scheduling.

In the Internet, the concepts of Application Oriented Networking (AON) [9] and Application-driven Networking [48, 35] have been being explored to enable coordination among disparate applications, to enforce application-specific policies, to improve visibility of information flow, and to enhance application optimization and QoS. While these concepts are generic enough to be applied to wireless sensor networks, the techniques employed in and the problems faced by the Internet are quite different from those in sensor networks, due to the differences in both technologies and application domains. For instance, the extreme resource (e.g., computation, communication, and energy) constraints are unique to sensor networks and are not the major issues in the Internet.

For customized resource provisioning to different applications, Darwin [23] has been proposed for run-time resource management in the Internet. To match application requirements to communication protocols, DANCE [79] has been proposed to provide a service-oriented view to communication services, thus to avoid the inflexibility that results from a fixed binding between an application and a specific protocol stack. Active networks [88] have also been proposed to enable functionalities such as application-specific multicast, information fusion, and other services leveraging network-based computing and storage.

## 7.2 Link estimation and routing

Link properties in 802.11b mesh networks and dense wireless sensor networks have been well studied in [10], [56], and [107]. They have observed that wireless links assume complex properties, such as wide-range non-uniform packet delivery rate at different distances, loose correlation between distance and packet delivery rate, link asymmetry, and temporal variations. Our study on link properties complements existing works by focusing on the differences between broadcast and unicast link properties, as well as the impact of interference pattern on the differences.

Differences between broadcast and unicast and their impact on the performance of AODV have been discussed in [66] and [22]. Our work complements [66] and [22] by experimentally studying the differences as well as the impact of environment, distance, and interference pattern on the differences, which were not the focus of [66] and [22]. [22] mentioned the difficulty of getting MAC feedback and thus focused on the method of beacon-based link estimation. Our work complements [22] by developing techniques for reliably fetching MAC feedback, which build the foundation for data-driven link estimation and routing. To improve the performance of AODV, [66] and [22] also discussed reliability-based mechanisms (e.g., RSSI- and SNR-based ones) for blacklisting bad links. Since it has been shown that reliability-based blacklisting does not perform as well as ETX [40, 27, 95], we do not directly compare LOF to [66] and [22], instead we compare LOF to ETX.

Recently, great progress has been made regarding routing in wireless sensor networks as well as in mesh networks. Routing metrics such as ETX [27, 95] and ETT/WCETT [31] have been proposed and shown to perform well in real-world wireless networks [30]. The geography-based metric PRD [83] has also been proposed for energy-efficient routing in wireless sensor networks. Nevertheless, unicast link properties were still estimated using broadcast beacons in these works. Our work differs from existing approaches by experimentally demonstrating the difficulty of precisely estimating unicast link properties via those of broadcast beacons, and proposing the data-driven protocol LOF where unicast link properties are estimated via the data traffic itself.

Similar to LOF, SPEED [42] also used MAC latency and geographic information in route selection. In parallel with our work, [60] proposed NADV which also uses information from MAC layer. While focusing on real-time packet delivery and a general framework for geographic routing, [42] and [60] did not focus on the protocol design issues in data-driven link estimation and routing. They did not study the differences between broadcast and unicast link properties. They did not consider the importance of appropriate *exploratory neighbor sampling* either. SPEED switches next-hop forwarders after every packet transmission (as in L-se), and NADV does not perform exploratory neighbor sampling (as in L-ns), both of which degenerate network performance as shown in Section 3.5. Complementary to SPEED and NADV, moreover, we have analyzed the small sample size requirement in LOF, which shows the feasibility of data-driven link estimation. While [42] and [60] have evaluated their methods via simulation, we have studied the systems issues in reliably fetching MAC feedback and evaluated LOF via experiments in real networks with realistic traffic trace. Finally, [60] did not compare the performance of NADV with that of ETX and PRD.

The problem of local minimum or geographic void has been dealt with in routing protocols such as GPSR [53]. In this dissertation, therefore, we have not considered this problem since it is independent of our major concerns — data-driven link estimation and routing. As a part of our future work, we plan to incorporate techniques of dealing with geographic void into LOF, by adapting the definition of “effective geographic progress” (in Section 3.3.1) and routing around void. The impact of localization errors on geographic routing has been studied in [82]. In LOF, we adopted a separate software component that fine tunes the GPS readings to reduce localization inaccuracy, as also used in the field experiment ExScal [12].

## 7.3 Packing-oriented scheduling

In the past years, query processing in sensor networks has drawn a lot of attention [68, 97, 69, 72, 29, 28, 63, 43]. TinyDB [68] and Cougar [97] are two exemplary sensor network database systems which regard data collection as a database query process and then design mechanisms (such as semantic query forwarding and in-network aggregation) to execute the query efficiently. Mechanisms for efficient and robust in-network aggregation for query processing have also been proposed in [69] and [72]. Nonetheless, existing work in sensornet query processing has not focused on the QoS requirement of different applications, nor did they focus on the generic application-adaptive messaging in sensornets.

Unlike the query-oriented data modeling and data processing, another aspect in modeling data in sensor networks is to investigate the correlation among them and then take advantage of the correlation in reducing the cost of data collection. [57, 76, 75]. To this end, [57], [76], and [75] studied the problem of finding the best aggregation tree given the data sources and the correlation structure between the data sources. Our work complements [57], [76], and [75] by not assuming a priori knowledge of the data sources and their correlation, such that the algorithms are more generically applicable. Also, we study the general architecture for application-adaptive messaging, which is not the focus of the above works.

As a simple form of data aggregation, packet packing has been studied in [50] and [3], where several short packets are packed into a long data packet to reduce the overhead per bit of data delivered. It has been shown that packet packing can improve network throughput significantly. While focusing on IEEE 802.11-type networks of devices such as rechargeable laptops, energy consumption is not a focus of these works, and they did not study the problem of dynamically tune the packing policy as application QoS requirement changes. Nagle's algorithm [41] is also used in TCP to pack short data segments into longer ones, but it was not designed to be application-adaptive either.

## 7.4 Reliable and real-time data transport

The performance of packet delivery in dense sensor networks has been studied in [106]. The results show that, in the presence of heavy channel load, a commonly used loss recovery scheme at link layer (i.e., lost packets are retransmitted up to 3 times) does not mask packet loss, and more than 50% of the links observe 50% packet loss. The observation shows the challenge of reliable communication over multi-hop routes, since the reliability decreases exponentially as the number of hops increases.

The limitations of timers in TCP retransmission control have been studied in [105]. The author analyzes the intrinsic difficulties in using timers to achieve optimal performance and argues that additional mechanisms should be used. Despite its focus on TCP, the study also applies to retransmission control in sensor networks.

Block acknowledgment [20] has been proposed for error as well as flow control in the Internet. It considers the problem of in-order packet delivery. Therefore, a lost packet blocks the delivery of all the packets that are behind the lost one but have reached the receiver, as a result of which packet delivery delay is increased. Moreover, a sender can send packets at most up to its window size once a packet is sent but unacknowledged, thus the channel resource may be under-utilized.[23] Block acknowledgment also uses timers without addressing their limitations, which can render additional delay in packet delivery.

For packet-loss detection and retransmission control, DFRF [70] uses stop-and-wait implicit ack (SWIA). Yet DFRF does not address the issue of retransmission-incurred channel contention. Moreover, the retransmission timers in DFRF do not adapt to the varying ack-delay, which can introduce more retransmission or delay than necessary. To reduce the number of packet transmissions, DFRF uses raw data aggregation where multiple short packets are concatenated to form a longer packet. In the type of bursty convergecast as experienced by Lites and ExScal [12], it is more difficult to perform data aggregation at the mote level because motes detecting an event can be multiple hops away from one another and the length of a single sensor data entry is more than half of the packet length. Therefore, the current implementation of RBC is based on the paradigm of implicit-ack to reduce the number of packets competing for channel access. On the other hand, we believe that the methodologies developed in RBC (e.g., window-less block acknowledgment and differentiated contention control) can also be applied when there is data aggregation, in which case we can use explicit-ack packets to send out control information. The detailed study on this is a part of our future work.

RMST [86] and PSFQ [89] have shown the importance of hop-by-hop packet recovery in sensor networks. Yet RMST and PSFQ do not focus on bursty convergecast. Therefore, they do not cope with retransmission-incurred channel contention, they do not design mechanisms to alleviate delay incurred by retransmission timers (whose timeout values

[23]The situation is worsened by the fact that the window size is less than half of the buffer size in block acknowledgment and the fact that the buffer size is usually small (e.g., 16) in wireless sensor networks due to limited RAM.

are conservatively chosen to reduce unnecessary retransmissions), and they do not design mechanisms to reduce the probability of ack-loss. Our work complements theirs by identifying issues with existing hop-by-hop mechanisms in bursty convergecast and proposing approaches to address the issues.

CODA [90] and ESRT [81] have studied congestion control in sensor networks. They consider a traffic model where multiple sources are continuously or periodically generating packets. Therefore, they do not focus on real-time packet delivery in bursty convergecast, and they do not consider retransmission-incurred delay as well as channel contention. Recent work in [47] and [33] has studied different techniques for mitigating congestion and guaranteeing fairness in wireless sensor networks. Our work complements theirs by focusing on retransmission-based error control and retransmission scheduling. Several transport protocols, such as ATP [87] and WTCP [85], are also proposed for wireless ad hoc networks. Again, they do not face the challenges of reliable bursty convergecast.

## 7.5 Local stabilization

The concept of fault containment is proposed in [38], [18], and [73]. Nevertheless, [38] and [18] only consider fault containment for the major part of system states, which is not strict enough to guarantee that the amount of work (for example, the number of protocol actions executed) needed for a system to stabilize is a function of the perturbation size; [73] only considers the case where the impact of every fault is within constant distance from where it occurs, which is too strict to be applied to problems such as routing, where the locality of the problem[24] is not constant.

[24] We regard the locality of a problem as the maximum minimum distance between any two nodes that have to be involved in the definition of the problem.

A locally stabilizing protocol GS$^3$ is proposed in [99] for clustering as well as shortest path routing in wireless sensor networks where nodes are densely distributed. To achieve local stabilization, GS$^3$ takes advantage of the high node distribution density and the geographic information on node distribution; the sensor network model assumed by GS$^3$ makes it inapplicable to other general networks such as the Internet. In [38], algorithms are proposed to contain a single state-corruption during stabilization of a spanning tree, but these algorithms do not deal with multiple faults and the fail-stop of nodes.

In [18], a broadcast protocol is proposed to contain externally observable variables in the presence of state corruptions, but the protocol allows for global propagation of internal variables. In [39], a fault-containing self-stabilizing algorithm is proposed for a consensus problem, but it only considers linear network topologies and the distance faults propagate can be exponential in the perturbation size; and the algorithm does not apply to the problem of shortest path routing.

Loop-free distance-vector protocols DUAL [36] and LPA [37] are proposed for Internet routing. Nonetheless, neither DUAL nor LPA guarantees local stabilization: faults can propagate globally in DUAL as well as in LPA when local transient perturbations, such as congestion and state corruption, occur. This phenomenon becomes worse when networks are under stress, with transient faults happening more frequently for some period of time [92]. Furthermore, the time taken to break a loop which has already formed (e.g., due to state corruption) is not constant in DUAL and LPA; instead it is proportional to the length of the loop.

# CHAPTER 8

# CONCLUDING REMARKS

In this dissertation, we have studied the challenges that wireless communication, resource constraints, and application diversity pose to the architecture and protocol design of sensornet messaging. To address the challenges, we proposed the sensornet messaging architecture SMA that decomposes the messaging service into three components — traffic-adaptive link estimation and routing (TLR), application-adaptive structuring (AST), and application-adaptive scheduling (ASC) — at different levels of abstraction. For each component of the architecture (especially TLR and ASC), we proposed protocol(s) to perform its associated functionalities. More specifically, we proposed data-driven link estimation and routing to form the basic messaging structure in a traffic-adaptive manner, we proposed a distributed algorithm that schedules packet transmissions to improve the utility of in-network processing, and we proposed window-less block acknowledgment and differentiated contention control to provide reliable and real-time data transport. The architecture and the associated component instantiations have been verified via simulation, experimentation, and field sensornet deployments.

We have also studied the challenges that complex faults and large system scale pose to the design of fault-tolerant messaging protocols. To address the challenges, we proposed the concept of local stabilization that characterizes the desirable spatial and temporal properties of large scale fault-tolerant systems. For the basic messaging task of routing, we proposed a locally stabilizing protocol LSRP that guarantees local containment of fault impact and quick stabilization after fault occurrence. The properties of LSRP have been analytically proved, and its concepts have also been applied to building dependable sensornet messaging software.

**Future work.** Despite the fact that we and the community as a whole have made significant progress toward building the architecture and components for dependable messaging in sensornets, we are still at the infant stage of ubiquitous and dependable networked sensing. The ever-developing application domains and technologies will continuously challenge the design of systems services (including messaging) while bringing new opportunities to science, engineering, and our daily life.

In the context of messaging, there is still no consensus on what the right architecture for sensornets is, or whether we might ever have a unified architecture for such diversified application domains and systems technologies. We need to explore the broad options and verify the effectiveness of different messaging architectures through case studies in typical application domains. We also need to identify typical messaging patterns (e.g., convergecast, broadcast, and anycast) in sensornets, and study their impact on the design of messaging architecture.

Algorithmically speaking, there is still a rich set of foundational and systems issues. How to model a single wireless link and the whole dynamic sensornet systems? What are the capacity limits of different scheduling and structuring algorithms? How to provide differentiated QoS to different applications in resource constrained and dynamic sensornets? Answers to these questions will provide deeper understanding and better systems design for sensornet systems.

To fulfill their potential, sensornets need to be and are increasingly being integrated with existing pervasive systems (e.g., cell phones and sensor-rich vehicles) and the Internet. Among others, this trend raises questions in terms of both the architecture for the integrated systems and algorithmic design to guarantee QoS across the end-to-end systems. Answers to these questions will facilitate the application of networked sensing to help solve scientific, engineering, and societal issues in our world.

# APPENDIX A

# RBC: ACK-LOSS PROBABILITY IN RBC

For convenience, we define the following notations:

| | | |
|---|---|---|
| $p$ | : | the probability of losing a single (data) packet; |
| $N$ | : | the number of packets received in succession without any loss in the middle; |
| $N'$ | : | the number of packets lost in succession; |
| $B$ | : | the number of packets received in succession without any loss in the middle, after a packet is already received; |
| $A$ | : | the number of times that the acknowledgment for a packet is received at the sender. |

Assuming that packet losses are independent of one another, we have the probability mass functions for random variables $N$ and $N'$ as follows.

$$\begin{aligned} P[N = k] &= p(1-p)^k \\ P[N' = k] &= (1-p)p^k \end{aligned}$$

In RBC, when a packet $m$ is received at a receiver $R$, the acknowledgment for $m$ can reach back to the sender $S$ in two ways: $S$ snoops $m$ when it is forwarded by $R$ later, with probability $P_{self}$; or $S$ does not snoop $m$ but snoops a packet whose block acknowledgment acknowledges the reception of $m$, with probability $P_{ba}$. Therefore, the probability $P_{rbc}$ of

$S$ receiving the acknowledgment for $m$ can be derived as follows:

$$\begin{array}{lcl}
P_{self} & = & 1-p \\
P_{ba} & = & p\sum_{k=0}^{\infty} P[B=k]P[A \geq 1|B=k] \\
 & = & p\sum_{k=0}^{\infty} P[B=k](1-P[A=0|B=k]) \\
 & = & p\sum_{k=0}^{\infty} P[N=k+1](1-P[N'=k]) \\
 & = & \frac{p(1-3p+4p^2-2p^3)}{1-p+p^2} \\
P_{rbc} & = & P_{self}+P_{ba} \\
 & = & 1-p+\frac{p(1-3p+4p^2-2p^3)}{1-p+p^2}
\end{array}$$

Then, the probability $P'_{rbc}$ of losing the acknowledgment for a packet in RBC is $1-P_{rbc}$.

In the case of Lites trace and implicit-ack, $p = 22.7\%$. Therefore $P'_{rbc} = 8.89\%$, reducing the ack-loss probability of SWIA by a factor of 2.07.

# APPENDIX B

# RBC: PROBABILITY THAT AN ORPHAN PACKET HAS NOT BEEN RECEIVED

According to the analysis in Appendix A, if the probability of losing a single data packet is $p$, then the probability $P'_{rbc}$ of losing an ACK for a packet is calculated as follows:

$$P'_{rbc} = p - \frac{p(1-3p+4p^2-2p^3)}{1-p+p^2} \tag{B.1}$$

Thus, if a packet has been retransmitted $k$ ($k > 0$) times not due to NACK (but due to retransmission timer timeout), then the probability that the packet has been received by the receiver is $(P'_{rbc})^k$.

Given that the NACK for a lost packet is piggybacked onto a data packet only once, the probability $P_{nack}$ of losing a NACK for a lost packet is simply calculated as:

$$P_{nack} = p \tag{B.2}$$

Therefore, if an orphan packet has been retransmitted $k$ ($k \geq 0$) times not due to NACK, the probability $P_{loss}$ that the packet has not been received can be calculated as follows:

$$P_{loss} = \begin{cases} (1-(P'_{rbc})^k)\frac{P_{nack}}{P_{nack}+P'_{rbc}} & \text{if } k > 0 \\ \frac{P_{nack}}{P_{nack}+P'_{rbc}} & \text{if } k = 0 \end{cases}$$

Therefore,

$$P_{loss} = \begin{cases} (1-(P'_{rbc})^k)\frac{p}{p+P'_{rbc}} & \text{if } k > 0 \\ \frac{p}{p+P'_{rbc}} & \text{if } k = 0 \end{cases} \tag{B.3}$$

# APPENDIX C

# RBC: PROOF OF THEOREM 1

**Theorem 1 (Freedom of packet accumulation)** The orphan packets at a node do not accumulate indefinitely, as long as the packet loss rate along a link is less than 49.14%.

*Proof*: We first compute the probability $P_{orph}$ that an enqueued packet becomes an orphan packet. According to the definition of orphan packet, a packet becomes orphan when either of the following hold: it has been received, but the corresponding ACK is lost; it has been lost, but the corresponding NACK is lost. Therefore, we calculate $P_{orph}$ as follows:

$$P_{orph} = (1-p) \times P'_{rbc} + p \times P_{nack} \tag{C.1}$$

where $p$ is packet loss rate along a link.

To assure that orphan packets do not accumulate indefinitely even if fresh packets keep arriving, we only need to ensure that $P_{orph}$ is less than $P_{loss}$, the probability that an orphan packet will be transmitted.

From Formula B.3, we see that $P_{loss}$ is minimum when $k = 1$, that is, min$(P_{loss})$ = $\frac{(1-P'_{rbc})P_{nack}}{P_{nack}+P'_{rbc}}$. To guarantee that $P_{orph}$ is less than $P_{loss}$, we only need $P_{orph} <$ min$(P_{loss})$, that is,

$$(1-p) \times P'_{rbc} + p \times P_{nack} < \frac{(1-P'_{rbc})P_{nack}}{P_{nack}+P'_{rbc}}$$

Solving this inequality, we get $p < 49.14\%$.

More intuitively, Figure C.1 shows the relationship between $P_{orph}$ and $\min(P_{loss})$, as $p$

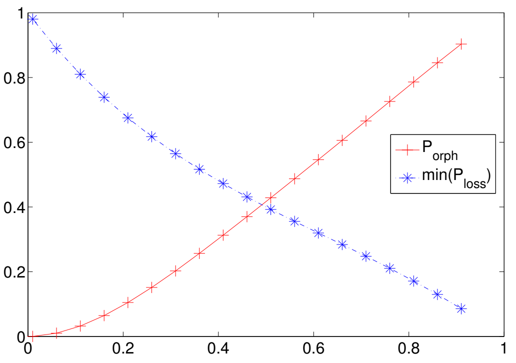


Figure C.1: $P_{orph}$ vs. $\min(P_{loss})$

changes.

$\square$

# APPENDIX D

# LSRP: PROOF OF LEMMA 1

In this appendix, we present the proof sketch for Lemma 1. Due to limitation of space, the complete proofs of all the theorems and lemmas in Chapter 6 are relegated to [16].

*Lemma 1: Starting at an arbitrary state $q_0$ where there is only one perturbed region, every system computation reaches a state in $\mathcal{L}$ within $O(P(q_0))$ time, and the range of contamination is $O(P(q_0))$.*

*Proof*: Let $p$ be the perturbation size $P(q_0)$ for the system at state $q_0$.

Let $R_p$ be the perturbed region.[25] The set of nodes that are not in $R_p$ is called the *healthy region* $R_h$[26]. A node $i$ in $R_p$ is called a *perturbed boundary node* if it has a neighbor $j$ that is in $R_h$, and node $j$ is called a *healthy boundary node* correspondingly. Given $R_p$ and $R_h$ at $q_0$, we let $P_d$ = $max\{hops(i, R_h) : i \in R_p\}$, where $hops(i, R_h) = min\{hops(i, j, G') : j \in R_h\}$ and $hops(i, j, G')$ = the number of hops in the shortest path between nodes $i$ and $j$ in $G'$.

There are three major variables (i.e., $d.i$, $p.i$, and $ghost.i$) maintained at each node $i$ in the system. According to the protocol, the corruption of $p.i$ at node $i$ does not contaminate any neighbor $j$ of $i$, therefore the corruption of $p.i$ at any node $i$ does not propagate.

[25]In shortest path routing, the number of nodes in the perturbed region is equal to $P(q_0)$.

[26]The subgraph of $G$ on $R_h$ can be disconnected.

However, the corruption of $d.i$ or $ghost.i$ at $i$ can contaminate the neighbors of $i$, thus it can propagate. Since the corruption of $d.i$ is more complicated (we can see this later in the proof) than that of $ghost.i$, we use it as the major theme leading the proof here. (Moreover, the effect of the corruption of mirror variables is the same as the corruption of $ghost.i$, except that the former adds constant factor $I_{syn}$ to the asymptotic expression of the theorems in Chapter 6. Thus, for simplicity of presentation, we skip it here.)

When a single perturbed region occurs in the system, there are two cases: (i) the "distance value" of every perturbed node is no less than what all the healthy boundary nodes can offer, i.e., $(\forall i, j : i \in R_p \wedge j$ is a healthy boundary node $\Rightarrow d.i \geq d.j + \min\{w.j.k : k \in R_p \wedge (j,k) \in E'\})$; (ii) the "distance value" of some or all perturbed nodes is less than what some or all healthy boundary nodes can offer. Due to limitation of space, here we only present the proof sketch for case (i). (The proof method for case (ii) is similar, and the proof is relegated to [16].)

For case (i), here we only consider the subcase where there exists at least one perturbed node $l$ where $d.l$ is corrupted at state $q_0$. Then, it is possible that there exists no healthy boundary node that is a source of fault propagation at state $q_0$; it is also possible that there exist some healthy boundary nodes that are sources of fault propagation at $q_0$. Due to limitation of space, we only analyze the latter case here.

When there are some non-root healthy boundary nodes that are sources of fault propagation at state $q_0$, containment waves can be initiated at those nodes, and the perturbed region $R_p$ can expand outward to the healthy region $R_h$. But this expansion is bounded. Also, here we only consider the case where there is only one such non-root healthy boundary node $i$ that is a source of fault propagation at $q_0$ (see Figure D.1). Then, the parent of $i$ must be perturbed at $q_0$ (i.e., $p.i \in R_p$) because otherwise node $i$ would not be a source

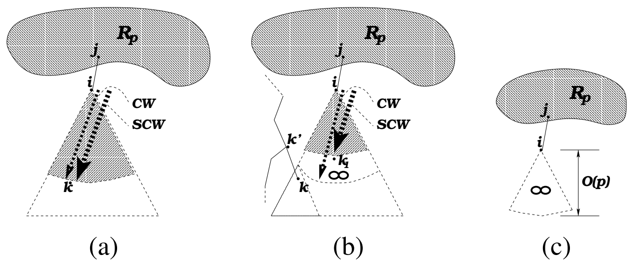


Figure D.1: The "distance value" of every perturbed node is no less than what all the healthy boundary nodes can offer, and there are some sources of fault propagation. In the figure, $SW$, $CW$, and $SCW$ stand for stabilization wave, containment wave, and super-containment wave respectively.

of fault propagation. Action $C_1$ is the only action that is enabled at node $i$ at $q_0$, and the execution of $C_1$ at $i$ will initiate a containment wave that is propagated toward the subtree rooted at node $i$, by the execution of action $C_1$ at every node involved in this containment wave. Thus, the perturbed region $R_p$ expands outward to the healthy region $R_h$. However, this containment wave will not propagate unbounded because of the following facts:

(I) If action $C_2$ has not been executed at node $i$ before the moment that is one round after all the nodes in $S$, the original set of perturbed nodes at state $q_0$, have converged to a legitimate state (see Figure D.1(a)(b)), the farthest distance (in terms of hops) the containment wave can propagate is $O(p)$, and the system reaches a state in $\mathcal{L}$ in $O(p)$ amount of time. The reasoning is as follows.

   (a) Starting at state $q_0$, it takes $O(p)$ amount of time for all the nodes in $S$ to converge to a legitimate state.

   We first show that starting at state $q_0$, the system reaches a state $q_0'$where $(\forall k : k \in V' \Rightarrow \neg ghost.i \wedge \neg SP.k)$ holds within $O(p)$ amount of time. The reason is as follows. Because the "distance value" of every perturbed node is no less than that of all healthy boundary nodes, no action except $C_1$ can be enabled at a healthy boundary node $i'$ that is not a source of fault propagation at $q_0$, but the execution of $C_1$ at $i'$ can only add $i'$ to the containment tree rooted at another healthy boundary node that is a source of fault propagation at state $q_0$ (in this case, it is node $i$). As for node $i$, no action except $C_1$ can be enabled at it. But the execution of $C_1$ at $i$ only adds node $i$ to a containment tree rooted at itself. Before action $C_2$ is executed at node $i$, actions $S_2$ that can set $i$ as the parent of its neighbor $j$ in $S$ will not be executed at $j$ because $ghost.i = true$. Therefore, no node in $S$ will be involved in the containment wave initiated at $i$, and actions executed at nodes in $(V \setminus S)$ will not affect actions executed at nodes in $S$ in terms of the growing or shrinking of containment trees in $S$. Thus, the proof follows that for Theorem 2 too. As shown in the proof for Theorem 2, the number of non-empty containment trees reaches 0 within $O(p)$ amount of time. When all the containment trees are empty, the system is at a state $q'_0$ where $(\forall k : k \in V' \Rightarrow \neg ghost.i \wedge \neg SP.k)$ holds.

We can see that, starting at state $q'_0$, the set of nodes in $S$ converge to a legitimate state within $O(p_d)$ amount of time. Let us denote the system state at this moment as state $q_1$.

Therefore, starting at state $q_0$, the set of nodes in $S$ converge to a legitimate state within $O(p) + O(p_d) = O(p)$ amount of time.

(b) After all the nodes in $S$ converge to a legitimate state $q_1$, the system stabilizes to a state in $\mathcal{L}$ within $O(p)$ amount of time.

When all the nodes in $S$ converge to a legitimate state $q_1$, and action $C_2$ has not been executed at node $i$, action $S_2$ will be executed at $i$. And a super-containment wave will propagate from $i$ downward to nodes in the containment tree $CT_i$, by the execution of action $SC$ at those nodes. There are two possible cases here:

- Action $C_2$ has not been executed at any node $j$ in $CT_i$ before the super-containment wave arrives at it (see Figure D.1(a)).

  In this case, because the super-containment wave propagates faster than the containment wave, it will catch up with the containment wave, and sets the state of every node $l$ in $CT_i$ to a legitimate one where $\neg ghost.l \wedge LH.l$ holds.

  Because the propagation speeds of different diffusing waves are controlled by introducing delays in action execution when a node schedules its enabled actions, and there is difference between clock rates at different nodes, $d_c > \alpha \cdot (d_{sc} + U)$ should hold in order for the super-containment wave to catch up with the containment wave. Under this condition, let $p_{cw}$ be the maximum distance (in terms of the number of hops) that the containment wave can propagate before the super-containment wave catches up with it. Then $(U + d_{sc})p_{cw} + O(p) = (L + d_c)p_{cw}$ holds, which implies that $p_{cw} = \frac{O(p)}{d_c - d_{sc} + L - U} = O(p)$.

  Therefore, the range of contamination and the time taken for the system to stabilize to a state in $\mathcal{L}$ is $O(p)$ in this case.

- Action $C_2$ has already been executed at some node $k_1$ in $CT_i$ before the super-containment wave arrives at it (see Figure D.1(b)).

In two cases will this happen: first, before the super-containment wave arrives at $k_1$, the containment wave has reached the leaf nodes of the subtree rooted at $k_1$ in $CT_i$; second, there exists a node $k$ in the subtree rooted at $k_1$ in $CT_i$ such that $(\exists k' : k' \in N.k \wedge k' \notin CT_i \wedge PS.k.k')$.

It is easy to see that in either case, the range of contamination is no more than that in the case where action $C_2$ has not been executed at any node $j$ in $CT_i$ before the super-containment wave arrives at it. Therefore, the range of contamination is still $O(p)$.

For any node $l$ in $CT_i$ where action $C_2$ has been executed before the super-containment wave reaches it, action $S_2$ will be executed at it within time proportional to $min\{hops(l, m, G) : m \in CT_i \wedge$ the super-containment wave has reached node $m\}$, which is $O(p)$. And the execution of $S_2$ at $l$ will set $LH.l$ to $true$.

We can see that the system stabilizes to a state in $\mathcal{L}$ within time $O(p)$ in this case.

(II) If action $C_2$ has been executed at node $i$ before the moment that is one round after all the nodes in $S$ (i.e., the original set of perturbed nodes at state $q_0$) have converged to a legitimate state (see Figure D.1(c)), the farthest distance the containment wave can propagate is still $O(p)$, and the system reaches a state in $\mathcal{L}$ in $O(p)$ amount of time. The reasoning is as follows.

It is easy to see that the depth of the containment tree $CT_i$ is $O(p)$ in this case. Thus the range of contamination is also $O(p)$.

After the execution of $C_2$ at node $i$, $d.l = \infty$ for every node $l$ that was in $CT_i$ once and has executed $C_2$. Let us denote the set of these nodes as $S'$. It is possible for some nodes in $S'$ to be added to the containment tree of some node $j'$ in $S$, but this can increase the depth of $CT_{j'}$ by at most $O(p)$. Therefore, the system will still reach a state $q_0'$ where all the containment trees are empty within time $O(p)$. And the same as analyzed before, the system will reach a state in $\mathcal{L}$ within time $O(p)$, starting at state $q_0'$. Therefore, starting at state $q_0'$, the system reaches a state in $\mathcal{L}$ within time $O(p)$.

$\square$

# BIBLIOGRAPHY

[1] Crossbow technology inc. http://www.xbow.com/.

[2] EmStar: Software for wireless sensor networks. http://cvs.cens.ucla.edu/emstar/.

[3] Karlnet's turbocell: Enhancing the capabilities of standard 802.11. http://www.karlnet.com/Documents/turbocellwhitepaper.zip.

[4] Rmase. http://www2.parc.com/isl/groups/era/nest/Rmase/default.html.

[5] Tinyos. http://www.tinyos.net/.

[6] Tmote sky sensor node. http://www.moteiv.com/.

[7] Wireless embedded systems. *http://webs.cs.berkeley.edu*.

[8] A Lites event traffic trace. http://www.cse.ohio-state.edu/∼zhangho/publications/Lites-trace.txt, 2003.

[9] Application-oriented networking. http://www.cisco.com/, 2005.

[10] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. Link-level measurements from an 802.11b mesh network. In *ACM SIGCOMM*, pages 121–132, 2004.

[11] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks (Elsevier)*, 38(4):393–422, 2002.

[12] A. Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, V. Naik, V. Kulathumani, Hongwei Zhang, H. Cao, M. Sridhara, S. Kumar, N. Seddon, C. Anderson, T. Herman, N. Trivedi, C. Zhang, M. Gouda, Y. R. Choi, M. Nesterenko, R. Shah, S. Kulkarni, M. Aramugam, L. Wang, D. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker. Exscal: Elements of an extrem scale wireless sensor network. In *RTCSA*, 2005.

[13] Anish Arora, Emre Ertin, Rajiv Ramnath, Mikhail Nesterenko, and William Leal. Kansei: A high-fidelity sensing testbed. *IEEE Internet Computing*, March 2006.

[14] Anish Arora and et al. A Line in the Sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks (Elsevier)*, 46(5), 2004.

[15] Anish Arora and Hongwei Zhang. Lsrp: Local stabilization in shortest path routing. *IEEE/ACM Transactions on Networking*, June, 2006.

[16] Anish Arora and Hongwei Zhang. Local stabilization in shortest path routing. *Technical Report, OSU-CISRC-7/03-TR45, The Ohio State University (ftp://ftp.cis.ohio-state.edu/pub/tech-report/2003/TR45.ps)*, July, 2003.

[17] Baruch Awerbuch, David Holmer, and Herbert Rubens. High throughput route selection in multi-rate ad hoc wireless networks. Technical report, Johns Hopkins University, 2003.

[18] Yossi Azar, Shay Kutten, and Boaz Patt-Shamir. Distributed error confinement. In *ACM PODC*, pages 33–42, 2003.

[19] Sandip Bapat, Vinod Kulathumani, and Anish Arora. Analyzing the yield of exscal, a large-scale wireless sensor network experiment. In *IEEE ICNP*, pages 53–62, 2005.

[20] G. Brown, M. Gouda, and R. Miller. Block acknowledgment: Redesigning the window protocol. In *ACM SIGCOMM*, pages 128–134, 1989.

[21] Alberto Cerpa, Jennifer Wong, Miodrag Potkonjak, and Deborah Estrin. Temporal properties of low power wireless links: Modeling and implications on multi-hop routing. In *ACM MobiHoc*, pages 414–425, 2005.

[22] Ian Chakeres and Elizabeth Belding-Royer. The utility of hello messages for determining link connectivity. In *WPMC*, 2002.

[23] Prashant Chandra, Yang hua Chu, Allan Fisher, Jun Gao, Corey Kosak, T.S. Eugene Ng, Peter Steenkiste, Eduardo Takahashi, and Hui Zhang. Darwin: Customizable resource management for value-added network services. In *ICNP*, 1998.

[24] Young-Ri Choi, Mohamed Gouda, Hongwei Zhang, and Anish Arora. Stabilization of grid routing in sensor networks. *AIAA Journal of Aerospace Computing, Information, and Communication*, to appear.

[25] Maurice Chu, Horst Haussecker, and Feng Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. Technical Report P2001-10113, PARC, 2001.

[26] David Clark. The design philosophy of the DARPA internet protocols. In *ACM SIGCOMM*, 1988.

[27] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *ACM MobiCom*, pages 134–146, 2003.

[28] Amol Deshpande, Carlos Guestrin, Wei Hong, and Samuel Madden. Exploiting correlated attributes in acquisitional query processing. Technical report, Intel Research - Berkeley, 2004.

[29] Amol Deshpande, Carlos Guestrin, Samuel Madden, Joseph Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[30] Richard Draves, Jitendra Padhye, and Brian Zill. Comparison of routing metrics for static multi-hop wireless networks. In *ACM SIGCOMM*, pages 133–144, 2004.

[31] Richard Draves, Jitendra Padhye, and Brian Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *ACM MobiCom*, pages 114–128, 2004.

[32] Cheng Tien Ee and Ruzena Bajcsy. Congestion control and fairness for many-to-one routing in sensor networks. In *ACM SenSys*, pages 148–161, 2004.

[33] Cheng Tien Ee and Ruzena Bajcsy. Congestion control and fairness for many-to-one routing in sensor networks. In *ACM SenSys*, pages 134–147, 2004.

[34] Kai-Wei Fan, Sha Liu, and Prasun Sinha. On the potential of structure-free data aggregation in sensor networks. In *IEEE INFOCOM*, 2006.

[35] Jonathan Follows and Detlef Straeten. *Application-Driven Networking: Concepts and Architecture for Policy-based Systems*. IBM, December 1999.

[36] J. J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Transactions on Networking*, 1(1):130–141, 1993.

[37] J. J. Garcia-Lunes-Aceves and Shree Murthy. A path-finding algorithm for loop-free routing. *IEEE/ACM Transactions on Networking*, 5(1):148–160, 1997.

[38] Sukumar Ghosh, Arobinda Gupta, Ted Herman, and Sriram V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *ACM PODC*, pages 45–54, 1996.

[39] Sukumar Ghosh and Xin He. Scalable self-stabilization. In *IEEE ICDCS'WSS*, pages 18–24, 1999.

[40] Omprakash Gnawali, Mark Yarvis, John Heidemann, and Ramesh Govindan. Interaction of retransmission, blacklisting, and routing metrics for reliability in sensor network routing. In *IEEE SECON*, pages 34–43, 2004.

[41] Mohamed G. Gouda. *Elements of Network Protocol Design*. John Wiley and Sons, 1998.

[42] Tian He, John Stankovic, Chenyang Lu, and Tarek Abdelzaher. SPEED: A stateless protocol for real-time communication in sensor networks. In *IEEE ICDCS*, 2003.

[43] Joseph Hellerstein, Wei Hong, Samuel Madden, and Kyle Stanek. Beyond average: Toward sophisticated sensing with queries. In *IPSN*, 2003.

[44] Frederick Hillier and Gerald Lieberman. *Introduction to Operations Research*. McGraw-Hill, 2001.

[45] Myles Hollander. *Nonparametric statistical methods*. Wiley, 1999.

[46] Christian Huitema. *Routing in the Internet*. Prentice-Hall, Inc., 1999.

[47] Bret Hull, Kyle Jamieson, and Hari Balakrishnan. Mitigating congestion in wireless sensor networks. In *ACM SenSys*, pages 134–147, 2004.

[48] IBM. *Application Driven Networking: Class of Service in IP, Ethernet and ATM Networks*. IBM, August 1999.

[49] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM*, pages 314–329, 1988.

[50] Ashish Jain, Marco Gruteser, Mike Neufeld, and Dirk Grunwald. Benefits of packet aggregation in ad-hoc wireless network. Technical Report CU-CS-960-03, University of Colorado at Boulder, August 2003.

[51] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., 1991.

[52] Mahesh Jayaram and George Varghese. Crash failures can drive protocols to arbitrary states. In *ACM PODC*, pages 247 – 256, 1996.

[53] Brad Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *ACM MobiCom*, pages 243–254, 2000.

[54] Vikas Kawadia and P. R. Kumar. Principles and protocols for power control in ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 23(5):76–88, 2005.

[55] Almudena Konrad, Ben Zhao, and Anthony Joseph. A markov-based channel model algorithm for wireless networks. *Wireless Networks*, 9:189–199, 2003.

[56] David Kotz, Calvin Newport, and Chip Elliott. The mistaken axioms of wireless-network research. Technical Report TR2003-467, Dartmouth College, Computer Science, July 2003.

[57] Rajnish Kumar, Matthew Wolenetz, Bikash Agarwalla, JunSuk Shin, Phillip Hutto, Arnab Paul, and Umakishore Ramachandran. DFuse: A framework for distributed data fusion. In *ACM SenSys*, 2003.

[58] Craig Labovitz, Ahba Ahuja, Roger Wattenhofer, and Srinivasan Venkatachary. The impact of internet policy and topology on delayed routing convergence. In *IEEE INFOCOM*, pages 537–546, 2001.

[59] Craig Labovitz, G. Robert Malan, and Farnam Jahanian. Origins of internet routing instability. In *IEEE INFOCOM*, pages 218–226, 1999.

[60] Seungjoon Lee, Bobby Bhattacharjee, and Suman Banerjee. Efficient geographic routing in multihop wireless networks. In *ACM MobiHoc*, pages 230–241, 2005.

[61] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *ACM SenSys*, pages 126–137, 2003.

[62] J. Li, C. Blake, D.S.J. De Couto, H.I. Lee, and R. Morris. Capacity of ad hoc wireless networks. In *ACM MobiCom*, pages 61–69, 2001.

[63] Xin Li, Young Jin Kim, Ramesh Govindan, and Wei Hong. Multi-dimensional range queries in sensor networks. In *ACM SenSys*, 2003.

[64] Juan Liu, Feng Zhao, and Dragan Petrovic. Information-directed routing in ad hoc sensor networks. In *ACM WSNA*, 2003.

[65] Ting Liu and Margaret Martonosi. Impala: A middleware system for managing autonomic, parallel sensor systems. In *ACM PPoPP*, 2003.

[66] Henrik Lundgren, Erik Nordstrom, and Christian Tschudin. Coping with communication gray zones in ieee 802.11b based ad hoc networks. In *ACM WoWMoM*, pages 49–55, 2002.

[67] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996.

[68] Samuel Madden, Michael Franklin, and Joseph Hellerstein. Tinydb: An acquisitional query processing system for sensor systems. In *ACM Transactions on Database Systems*, 2004.

[69] Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *OSDI*, 2002.

[70] M. Maroti. The directed flood routing framework. In *Technical report, Vanderbilt University, ISIS-04-502*, 2004.

[71] Klara Nahrstedt, Dongyan Xu, Duangdao Wichadakul, and Baochun Li. Qos-aware middleware for ubiquitous and heterogeneous environments. *IEEE Communications Magazine*, 2001.

[72] Suman Nath, Phillip Gibbons, Srinivasan Seshan, and Zachary Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *ACM SenSys*, 2004.

[73] Mikhail Nesterenko and Anish Arora. Local tolerance to unbounded byzantine faults. In *IEEE SRDS*, pages 22–31, 2002.

[74] Charles E. Perkins. *Ad Hoc Networking*. Addison Wesley, 2001.

[75] Dragan Petrove, Rahul Shah, Kannan Ramchandran, and Jan Rabaey. Data funneling: Routing with aggregation and compression for wireless sensor networks. In *ICC Worshops*, 2003.

[76] Peter Pietzuch. Path optimization in stream-based overlay networks. Technical Report TR-26-04, Harvard University, 2004.

[77] Joseph Polastre, Jonathan Hui, Philip Levis, Jerry Zhao, David Culler, Scott Shenker, and Ion Stoica. A unifying link abstraction for wireless sensor networks. In *ACM SenSys*, pages 76–89, 2005.

[78] Joseph Polatre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *ACM SenSys*, 2004.

[79] Bernd Reuther, Dirk Henrici, and Markus Hillenbrand. DANCE: Dynamic application oriented network services. In *EUROMICRO*, 2004.

[80] Loren Rittle, Venu Vasudevan, Nitya Narasimhan, and Chen Jia. MUSE: Middleware for using sensors effectively. In *INSS*, 2005.

[81] Y. Sankarasubramaniam, O. Akan, and I. Akyildiz. ESRT: Event-to-sink reliable transport in wireless sensor networks. In *ACM MobiHoc*, pages 177–188, 2003.

[82] Karim Seada, Ahmed Helmy, and Ramesh Govindan. On the effect of localization errors on geographic face routing in sensor networks. In *IEEE-ACM IPSN*, 2004.

[83] Karim Seada, Marco Zuniga, Ahmed Helmy, and Bhaskar Krishnamacari. Energy-efficient forwarding strategies for geographic routing in lossy wireless sensor networks. In *ACM SenSys*, 2004.

[84] Aman Shaikh, Lampros Kalampoukas, Rohit Dube, and Anujan Varma. Routing stability in congested networks: Experimentation and analysis. In *ACM SIGCOMM*, pages 163–174, 2000.

[85] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. WTCP: a reliable transport protocol for wireless wide-area networks. In *ACM MobiCom*, pages 231–241, 1999.

[86] F. Stann and J. Heidemann. RMST: Reliable data transport in sensor networks. In *IEEE SNPA*, pages 102–112, 2003.

[87] K. Sundaresan, V. Anantharaman, H.Y. Hsieh, and R. Sivakumar. ATP: A reliable transport protocol for ad-hoc networks. In *ACM MobiHoc*, pages 64–75, 2003.

[88] David L. Tennenhouse and David J. Wetherall. Towards an ative network architecture. *Computer Communication Review*, 26(2), 1996.

[89] C.Y. Wan, A. Campbell, and L. Krishnamurthy. PSFQ: A reliable transport protocol for wireless sensor networks. In *ACM WSNA*, pages 1–11, 2002.

[90] C.Y. Wan, S. Eisenman, and A. Campbell. CODA: Congestion detection and avoidance in sensor networks. In *ACM SenSys*, pages 266–279, 2003.

[91] Hong Shen Wang and Nader Moayeri. Finite-state markov channel - a useful model for radio communication channels. *IEEE Transactions on Vehicular Technology*, 44(1):163–171, 1995.

[92] Lan Wang, Xiaoliang Zhao, Dan Pei, Randy Bush, Daniel Massey, Allison Mankin, S. Felix Wu, and Lixia Zhang. Observation and analysis of BGP behavior under stress. In *ACM SIGCOMM Internet Measurement Workshop*, pages 138–147, 2002.

[93] Andreas Willig. A new class of packet- and bit-level models for wireless channels. In *IEEE PIMRC*, 2002.

[94] Andreas Willig. A new class of packet- and bit-level models for wireless channels. In *IEEE PIMRC*, 2002.

[95] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *ACM SenSys*, pages 14–27, 2003.

[96] Alec Woo, Sam Madden, and Ramesh Govindan. Networking support for query processing in sensor networks. *Communications of the ACM*, 47(6):47–52, 2004.

[97] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. In *ACM SIGMOD*, 2002.

[98] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *IEEE InfoCom*, pages 1567–1576, 2002.

[99] Hongwei Zhang and Anish Arora. GS$^3$: Scalable self-configuration and self-healing in wireless sensor networks. *Computer Networks (Elsevier)*, 43(4):459–480, 2003.

[100] Hongwei Zhang, Anish Arora, Young Ri Choi, and Mohamed Gouda. Reliable bursty convergecast in wireless sensor networks. In *ACM MobiHoc*, 2005.

[101] Hongwei Zhang, Anish Arora, and Prasun Sinha. Learn on the fly: Quiescent routing in sensor network backbones. Technical Report OSU-CISRC-7/05-TR48, The Ohio State University (http://www.cse.ohio-state.edu/∼zhangho/publications/LOF-TR.pdf), 2005.

[102] Hongwei Zhang, Anish Arora, and Prasun Sinha. Learn on the fly: Data-driven link estimation and routing in sensor network backbones. In *IEEE INFOCOM*, 2006.

[103] Hongwei Zhang, Loren J. Rittle, and Anish Arora. Application-adaptive messaging in sensor networks. Technical Report OSU-CISRC-6/06-TR63, The Ohio State University, 2006.

[104] Hongwei Zhang, Lifeng Sang, and Anish Arora. Link estimation and routing in sensor networks: Beacon-based or data-driven? Technical Report OSU-CISRC-6/06-TR64, The Ohio State University, 2006.

[105] L. Zhang. Why TCP timers don't work well. In *ACM SIGCOMM*, pages 397–405, 1986.

[106] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *ACM SenSys*, pages 1–13, 2003.

[107] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *ACM SenSys*, pages 1–13, 2003.

# Index