Hierarchical Routing in Low-Power Wireless Networks

Konrad Iwanicki

June 2010


Copyright © 2010 by Konrad Stanislaw Iwanicki.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission of the author and/or a fee.

The university, company, product, and service names used in this dissertation are for identification purposes only. All trademarks and registered trademarks are the property of their respective owners.

The cover has been designed by the author.

The photo on the cover and in Figure 1.1 on page 21 has been reproduced by permission from Shana Farley, Crossbow Technology, Inc.


Some of the research presented in this dissertation was supported by grants from:

DEVLAB

development laboratories

vrije Universiteit amsterdam
Hierarchical Routing
in Low-Power Wireless Networks

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op dinsdag 29 juni 2010 om 15.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Konrad Stanisław Iwanicki

geboren te Radom, Polen
Abstract

POINT-TO-POINT routing is fundamental functionality in the Internet. Its objective is finding paths in the network along which any two devices (nodes) in different parts of the globe can exchange data with each other. It is the only functionality shared by all Internet nodes, be they in the Internet core or at its end-points, and conversely, to be a fully-fledged Internet node, any novel device has to implement this functionality.

Wireless sensor nodes are such novel devices that aim to extend the digital world of the Internet into the surrounding physical world. They are tiny and are equipped with sensors and actuators, so that, embedded in the surrounding environment, they can sense various features of the environment and respond to these features by actuating physical objects, thereby controlling the environment. To fulfill this task, the devices have to communicate with each other via low-power radios, forming so-called wireless sensor networks (abbreviated sensornets or WSNs), and also with other devices on the Internet. To this end, in principle they need point-to-point routing functionality. The objective of this dissertation is thus to provide a point-to-point routing protocol for sensornets.

Developing such a protocol, however, is extremely challenging. Due to their embedding in physical space, sensor nodes are severely constrained in terms of memory, bandwidth, and processing power. Moreover, the low-power wireless communication they employ is unreliable and does not allow for engineering node
connectivity easily. Finally, the nodes are typically deployed in large numbers to cover all required sensing and actuation points. These constraints lead to the following requirements for a point-to-point routing protocol for sensornets. First, the routing state nodes maintain in the protocol should be small as compared to the node population. Second, the routing paths the protocol finds should be close to the optimal possible ones, that is, the routing stretch of the protocol should be small. Third, the protocol should be robust against communication and node failures. Finally, it should be self-managed to a large extent. Fulfilling all these requirements simultaneously is extremely challenging.

This dissertation advocates hierarchical routing as a compelling point-to-point routing technique for low-power wireless embedded networks. The principal idea behind hierarchical routing is to organize nodes into a multi-level hierarchy of clusters. With such an organization, instead of maintaining routing state for every other node in the network, a node can maintain state only for a few clusters in its vicinity. In effect, routing state can be reduced tremendously, which is an important argument in favor of hierarchical routing. To date, however, there have been two major arguments against hierarchical routing, due to which this routing technique has been considered unappealing for sensornets. First, due to maintaining very small state, the stretch in hierarchical routing can be large in some network topologies. Second, the problem of organizing nodes into and maintaining a multi-level cluster hierarchy is extremely complex, and thus, implementing robust, self-managed hierarchical routing for resource-constrained sensor nodes may not be feasible. This dissertation debunks these arguments.

The contributions it makes are threefold. First, it presents PL-Gossip, a robust, self-managed hierarchical routing protocol for sensornets. PL-Gossip shows that hierarchy construction and maintenance can be performed with simple mechanisms, which require little state and can work on resource-constrained nodes in the presence of communication failures and even massive node failures. Second, the dissertation demonstrates experimentally that, although in arbitrary network topologies the stretch in hierarchical routing may be large, it is small in the topologies of sensornets, on average within 25–50% of the optimal one, depending on the hierarchy properties. This result is supported analytically and is essentially due to the geometric nature of sensornet topologies, resulting from embedding nodes in physical space. Third, the dissertation shows that, compared to other competing techniques, such as shortest-path routing, compact routing, and constant-state routing, hierarchical routing performs well in sensornets. For many applications, it arguably offers the best state-stretch trade-off. All in all, this dissertation enables hierarchical routing in sensornets and proves that hierarchical routing is a compelling technique for sensornets.
Looking at the cover of this dissertation, one sees only a single name. This reflects the fact that the road to becoming an independent researcher is, by and large, traveled in solitude. Nevertheless, my name as the sole one on the cover is also a huge simplification, because actually a number of people have supported me on the road to a PhD. I would like to use this opportunity to thank them.

First and foremost, I would like to thank my adviser, Maarten van Steen. While I could write endlessly how he taught me a great deal about authoring scientific papers, constantly infected me with his inexhaustible enthusiasm, and was making sure that I “had a life” besides research, I am most grateful for the independence of my research position. I had a total freedom of choosing my research topics and was able to occasionally disappear from Maarten’s agenda for a few months, just on a notice that I was working on an exciting problem and that he would get a paper draft when I was done. Even though such independence has had a huge impact on my development, few professors would risk offering it to their students.

Another great deal of thanks goes to Cezary Dubnicki and his Princeton team, especially Wojtek and Grzesiek. As a member of that team prior to my appointment in Amsterdam, I had an opportunity to learn how world-class research and development looks like and how to choose those research problems.
that can potentially have substantial practical impact.

I would have been neither in Amsterdam nor in Princeton but for my professors at Warsaw University, especially my Master’s adviser, Janina Mincer-Daszkiewicz. My time at Warsaw University laid solid foundations for my whole career.

Returning back to the dissertation, I would like to thank the members of my committee, professors: Maarten Boasson, Herbert Bos, Wan Fokkink, and Koen Langendoen. All of them dedicated a lot of time and effort to reviewing my research results, and their feedback helped to make important last-minute improvements to the dissertation.

Long before this dissertation was complete, a couple of other people helped me during my research. First of all, despite my being a PhD student, the DevLab company based in Eindhoven was willing to pay a proper salary for the most of my period in Amsterdam. I greatly appreciate that gesture, especially since our contact was rather sporadic. Tahir Azim from Stanford was a co-developer of the sensornet point-to-point routing library presented in Chapter 5 and Appendix C.2. Although we met in person only during a few sensornet conferences, our frequent tele-conferences were highly productive. My brother, Krzysztof Iwanicki, helped me with a demo application for the hierarchical routing library, which is mentioned in Appendix C.1 and was displayed on a number of occasions. His expertise in Java and computer graphics proved invaluable. Arno Bakker, a colleague from VU, not only was the translator of the Dutch summary of this dissertation, which can be found in Appendix D, but also many times offered his help when I was dealing with the Dutch bureaucracy. Unfortunately, because of a busy agenda, in all those years, I was unable to master the Dutch language sufficiently. Finally, my office-mate, Albana Gaba, enthusiastically shared the burden of deploying 50+ sensor nodes at the university. Without her superb womanly social skills, the deployment would have been far more exhausting. Those skills proved invaluable again when she helped arranging my PhD defense.

Since we have come to office-mates, together with Albana and my other office-mate, Vivek Rai, we were quite an interesting lot, occupying den P4.30. Each of us representing a different culture, we had surprisingly many topics to discuss. Our conversations and, more often, arguments were rarely within any bounds of political correctness, which was quite amusing considering other people in our environment. Vivek was also a formidable opponent in our never-ending tennis tournaments. I will miss those discussions and tennis matches a lot as they constituted a much-needed distraction from my work. My earlier short-term office mates, Michał, Swami, and Nick, as well as other colleagues from the university, which are too numerous to mention, have also left me some
nice memories.

Apart from my office-mates, keeping me sane was largely a task of my other friends. While many people came and went throughout my stay in Amsterdam, the following people could always be counted on: Agnieszka, András, Bálint, Ela, Iwona, Łukasz, Nives, Olek, and Paweł. I am grateful to them for all animated conversations, weekend activities, pints of beer, and bottles of other favorite Polish and Hungarian liquors.

I also must not forget about my family. Despite the distance that separated us, they provided constant support, and I was always waiting for the next time I would be able to visit them. Their attitude allowed me to have the luxury of studying abroad.

Last but not least, I would like to thank Kasia. It is hard to express how much I am grateful for her support. We shared the joy of having papers accepted and being granted awards, but also the sorrow over having papers rejected. Many of what I achieved was for her and because of her.

Konrad
Warsaw, Poland,
May 3, 2010
Abstract

Acknowledgments

1 Introduction

1.1 Wireless Sensor Networks

1.1.1 Immense Potential

1.1.2 Novel Challenges

1.2 Networking Low-Power Wireless Nodes

1.2.1 Decade of Research

1.2.2 Networking Fabric

1.3 Problem Statement

1.4 Thesis

1.5 Contributions and Outline

2 Background

2.1 Sensornet Applications

2.1.1 Examples of Application Domains

2.1.2 Common Features

2.2 Consequences for Routing

2.2.1 Hardware and Physical Limitations

2.2.2 Deployment Constraints

2.2.3 Putting Everything Together

2.3 Related Work on Routing

2.3.1 Shortest-Path Routing

2.3.2 Constant-State Routing

2.3.3 Compact Routing

2.3.4 Hierarchical Routing
### 3 Enabling Hierarchical Routing in Sensornets

#### 3.1 Routing Protocol Skeleton

#### 3.2 Basic Terms and Definitions

- **3.2.1 Area Hierarchy Model**
- **3.2.2 Hierarchical Routing Address**
- **3.2.3 Hierarchical Routing Table**

#### 3.3 Hierarchical Packet Forwarding

- **3.3.1 Forwarding Algorithm**
- **3.3.2 Remarks**

#### 3.4 Routing State Maintenance

- **3.4.1 Principal Idea**
- **3.4.2 Routing Table Maintenance**
- **3.4.3 Routing Address Maintenance**

#### 3.5 Link Quality Estimation

- **3.5.1 Measuring Link Quality**
- **3.5.2 Using Link Quality Measurements**

#### 3.6 Concluding Discussion

### 4 Evaluating Hierarchical Routing in Sensornets

#### 4.1 High-Level Simulation

- **4.1.1 Simulation Settings**
- **4.1.2 Routing State**
- **4.1.3 Routing Stretch**
- **4.1.4 Robustness and Self-Management**

#### 4.2 Implementing Hierarchical Routing

- **4.2.1 Sample Design Points**
- **4.2.2 Routing Protocol Scheme**
- **4.2.3 Design Point: Hierarchy Type**
- **4.2.4 Design Point: Propagating Hierarchy Information**
- **4.2.5 Implementation Summary**

#### 4.3 Low-Level Implementation-Based Simulation

- **4.3.1 Routing State**
- **4.3.2 Routing Stretch**
- **4.3.3 Comparison of Area and Landmark Hierarchies**
- **4.3.4 Robustness and Self-Management**
- **4.3.5 Comparison of Information Propagation Methods**

#### 4.4 Testbed Experiments

- **4.4.1 Micro-Benchmarks**
- **4.4.2 Long-Term Network Dynamics**
C Accompanying Software 261
   C.1 Hierarchical Routing Library . . . . . . . . . . . . . . 261
   C.2 General Routing Library . . . . . . . . . . . . . . . . . . . . 264

D Foreign-Language Summaries 265
   D.1 Samenvatting (Dutch Summary) . . . . . . . . . . . . . . 265
   D.2 Streszczenie (Polish Summary) . . . . . . . . . . . . . . . . 272
List of Figures

1.1 A sample wireless sensor node, Mica2Dot ........................................ 21
2.1 An overview of hardware components constituting a sensor node ........ 40
2.2 Technological divergence of sensor nodes from Internet routers ......... 49
2.3 The routing techniques spectrum w.r.t. the state-stretch trade-off ..... 62
3.1 Routing abstractions comprising a routing protocol ......................... 65
3.2 An example of a connected network ............................................... 66
3.3 An example of an area hierarchy in the network from Figure 3.2 ........ 67
3.4 The hierarchy from Figure 3.3 with central subclusters marked .......... 70
3.5 The label tree for the hierarchy from Figure 3.4 ............................. 71
3.6 The routing table of node P from Figure 3.4 .................................. 72
3.7 Routing a packet from node P.O.L from Figure 3.4 to node C.A.L .... 79
3.8 The stretch of the routing path from Figure 3.7 .............................. 80
3.9 A example of routing without involving cluster heads ...................... 81
3.10 A example of a label and a corresponding update vector ................ 89
3.11 A example of the label extension operation ................................. 90
3.12 A example of the label cut operation ........................................... 91
3.13 A example of the label combination operation ............................... 93
4.1 The cluster hierarchy height (high-level simulation) ......................... 111
4.2 The routing table size (high-level simulation) ................................. 112
4.3 The routing stretch (high-level simulation) .................................... 113
4.4 The hierarchy bootstrap latency (high-level simulation) .................... 115
4.5 Hierarchy bootstrap in PL-Gossip vs. an optimal centr. approach ...... 116
4.6 PL-Gossip under network dynamics (high-level simulation) ............. 118
4.7 The routing table size (TOSSIM) .................................................. 135
4.8 The routing stretch (TOSSIM) ..................................................... 138
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9</td>
<td>The routing state in area and landmark hierarchies (TOSSIM)</td>
</tr>
<tr>
<td>4.10</td>
<td>The routing stretch in area and landmark hierarchies (TOSSIM)</td>
</tr>
<tr>
<td>4.11</td>
<td>The hierarchy bootstrap latency (TOSSIM)</td>
</tr>
<tr>
<td>4.12</td>
<td>The bootstrap traffic (TOSSIM)</td>
</tr>
<tr>
<td>4.13</td>
<td>The stable-state maintenance traffic (TOSSIM)</td>
</tr>
<tr>
<td>4.14</td>
<td>The hierarchy recovery latency (TOSSIM)</td>
</tr>
<tr>
<td>4.15</td>
<td>The energy consumed with low-power listening (TOSSIM)</td>
</tr>
<tr>
<td>4.16</td>
<td>The energy efficiency of different solutions (TOSSIM)</td>
</tr>
<tr>
<td>4.17</td>
<td>A sample area hierarchy (testbed)</td>
</tr>
<tr>
<td>4.18</td>
<td>A sample landmark hierarchy (testbed)</td>
</tr>
<tr>
<td>4.19</td>
<td>A sample evolution of the pairwise internode reachability (testbed)</td>
</tr>
<tr>
<td>4.20</td>
<td>A Byzantine evolution of the internode reachability (testbed)</td>
</tr>
<tr>
<td>5.1</td>
<td>The routing state vs. the network size (TOSSIM)</td>
</tr>
<tr>
<td>5.2</td>
<td>The routing state vs. the network density (TOSSIM)</td>
</tr>
<tr>
<td>5.3</td>
<td>The routing stretch vs. the network size (TOSSIM)</td>
</tr>
<tr>
<td>5.4</td>
<td>The routing stretch vs. the network density (TOSSIM)</td>
</tr>
<tr>
<td>5.5</td>
<td>The stretch and state of BVR in various configurations (TOSSIM)</td>
</tr>
<tr>
<td>B.1</td>
<td>A TelosB node with the sensor suite (TMote Sky)</td>
</tr>
<tr>
<td>B.2</td>
<td>The node placement in our testbed</td>
</tr>
<tr>
<td>B.3</td>
<td>An RSSI matrix for the testbed (RF power of −25 dBm)</td>
</tr>
<tr>
<td>B.4</td>
<td>An LQI matrix for the testbed (RF power of −25 dBm)</td>
</tr>
<tr>
<td>B.5</td>
<td>A PLR matrix for the testbed (RF power of −25 dBm)</td>
</tr>
<tr>
<td>B.6</td>
<td>The testbed node degree distribution (RF power of −25 dBm)</td>
</tr>
<tr>
<td>B.7</td>
<td>The testbed path length distribution (RF power of −25 dBm)</td>
</tr>
<tr>
<td>C.1</td>
<td>A screenshot of the user interface of the visualization software</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Representative examples of MCUs suitable for sensornets . . . . . 41
2.2 Representative examples of radios suitable for sensornets . . . . . 44
3.1 Entries in the routing tables of sample nodes from Figure 3.3 . . . 74
4.1 Different hierarchy types in proposed protocols . . . . . . . . . . . 125
4.2 Different update propagation methods in proposed protocols . . . 130
4.3 The routing state and routing stretch (testbed) . . . . . . . . . . . 158
4.4 Comparison of different information propagation methods (testbed) 158
5.1 The code breakdown of routing protocols with static addresses . . 173
5.2 The performance of different routing techniques (two testbeds) . . 185
B.1 The main hardware components of the experimental testbed . . . . 252
B.2 The specification of a TelosB sensor node . . . . . . . . . . . . . . 253
B.3 The neighbor relation graph in the testbed depending on RF power . 259
List of Listings

3.1 The most important fields of a routing entry .................. 73
3.2 The routing packet header initialization function .......... 76
3.3 The next-hop lookup routing function for an area hierarchy ... 78
3.4 A highly simplified function for refreshing a routing entry .... 86
3.5 The label extension and label cut functions ................. 92
3.6 The label combine function .................................. 94
3.7 The function for synthesizing the hierarchy ............... 97
3.8 The function for recovering the hierarchy .................. 99
3.9 The function for initializing node routing state ............ 100
3.10 The function for periodically refreshing and broadcasting node state ... 101
3.11 The function for handling a heartbeat message reception ... 102

4.1 The next-hop lookup routing function for a landmark hierarchy .. 128
Gordon Bell has observed that a new class of computing appears and doubles approximately every decade — a phenomenon he labeled Bell’s Law of Computer Classes [7]. A new class of computing corresponds to some disruptive technology, which may be the consequence and combination of three novelties: a new platform, a new network, or a new interface with people or other information processing systems. There have been several computing classes to date, including supercomputers, mainframes, minicomputers, desktop workstations, wireless laptops, and mobile phones, most of which are still extensively used.

The idea to network together interconnected devices from such different classes into a single network of networks gave birth to the Internet a few decades ago [83]. That idea turned out to be a great success and since then the Internet has proliferated to become a global, ubiquitous communication infrastructure. Today, millions of computing devices, from supercomputers to mobile phones, networked by the Internet constitute a digital world. However, while this digital world is vital to the existence of many people and businesses, in many aspects it is...
still detached from the physical world around us. This is now starting to change.

A novel class of computing, which has the potential to finally bridge the gap between the digital world of the Internet and the surrounding physical world, is emerging: wireless sensor networks, collections of tiny devices integrating computing, wireless communication, and sensing \[2\] \[20\]. Their small size and unobtrusiveness imply that they can be embedded in the surrounding environment to sense and act upon various features of the environment at an unprecedented scale and fidelity.

1.1. WIRELESS SENSOR NETWORKS

Wireless sensor networks (abbr. sensornets or WSNs) are by all definitions a disruptive technology. While history evidences that a technology offering any of Bell’s aforementioned three novelties can already give birth to a new class of computing, sensornets offer all the three novelties simultaneously, by elegantly making use of the progress in miniaturization and energy efficiency. Their computing platform is based on miniature, ultra-low-power microcontrollers, capable of operating untethered for years, using just small batteries. Likewise, as their networking hardware, they adopt miniature, ultra-low-power radios, which eliminate obtrusive network wires and facilitate long unattended operation. Finally, they interface with the outside world through low-power sensors and actuators, which can collect information from and provide feedback to the surrounding environment. A sample sensor node is presented in Figure 1.1.

1.1.1. Immense Potential

The ability to deploy such tiny sensors in remote, uncontrolled, or hazardous environments to relay data about those environments to the Internet enables a plethora of novel applications. Biologists placed wireless sensor nodes in underground burrows of Leach’s Storm-Petrels, which are elusive birds breeding only on inaccessible islands \[93\]. For several weeks, the network deployed on Great Duck Island (Maine, USA) was providing invaluable data on the nesting habits of the birds at a scale unprecedented at that time \[123\]. Structural engineers deployed a network of vibration sensors on the Golden Gate Bridge in San Francisco (California, USA) to study how ambient factors, such as winds, affect the bridge \[62\]. Not only did the deployment give insight into those effects, but also demonstrated the potential of sensornets to provide continuous structural integrity monitoring. Geo-scientists are now running a three-year
1.1. WIRELESS SENSOR NETWORKS

Figure 1.1: A sample wireless sensor node, Mica2Dot. For computing, it adopts an ultra-low-power 8-bit microcontroller with the frequency up to 8 MHz, 4 KB of RAM for data, and 128 KB of program flash. For networking, it employs a short-range, low-power radio with the maximal bandwidth of 38.4 Kbits/s. For connecting to sensors and actuators, it offers 19 solderless expansion pins. The node can be powered by a small 3-V coin cell.

sensornet deployment on the hill tops of the Swiss Alps to investigate under which conditions permafrost causes avalanches and to study how current avalanche warning mechanisms can be improved [8]. Without the long, unattended operation and remote data collection ability offered by sensornets, such a study would be too laborious and hazardous to conduct.

The potential of sensornets, however, is much greater than just remote, unattended data collection. When coupled with an actuator, a sensor node can act upon readings of other sensors or commands from other devices on the Internet, effectively controlling its surrounding environment. Acoustic sensors deployed in a parking lot can detect when a car alarm is triggered and notify actuator nodes controlling security cameras about such an event [99]. The actuator node closest to the source of the alarm can then rotate its camera to verify whether a theft is taking place and to record the potential perpetrator. Air humidity and soil moisture sensors deployed in a field can wirelessly provide their readings to actuator nodes that control the irrigation system of the field [95]. In this way, the actuators can precisely administer water only to those areas of the field that require irrigation at a particular moment. Temperature sensors in an office building can normally relay their readings to actuators of the air-conditioning system, which controls the office climate. When fire erupts in the building, those
temperature sensors can verify the readings of smoke sensors and signal an alarm to another type of actuators: the actuators controlling ceiling lights. In response to the alarm, those actuators can collaboratively turn the lights on and off in an orderly manner to direct escaping people to the nearest safe emergency exit.

All in all, the ability of sensornets to monitor the environment, relay their observations to the Internet, receive feedback from the Internet, and act upon this feedback and the observations by controlling the environment can effectively blend the physical and the digital worlds. Consequently, sensornets have the potential to be the next milestone in the evolution of the Internet, where not only humans, but also everyday objects surrounding them communicate and interact with each other to collaboratively perform certain tasks, a vision often referred to as ubiquitous computing [135], cyber-physical systems [81], smart dust [134], or the Internet of things [133].

1.1.2. Novel Challenges

However, this immense potential of sensornets is accompanied by a number of challenges. The challenges stem mainly from the embedded nature of sensor devices, inherent in the above vision, notably their small form-factor and untethered, unattended operation. While the progress in miniaturization, governed by Moore’s law [97], generates a steady stream of smaller and smaller computing devices, the laws of physics limit the miniaturization of power sources: chemical energy density limits storage-based power sources, such as batteries, and ambient energy density limits sources utilizing renewable energy, such as solar cells. As a result, the form-factor and the period of unattended operation of a wireless sensor node are — and are likely to be in the future — determined by the power source used by the node. In other words, ensuring a long operation period of a node without increasing the form factor necessitates minimizing the energy consumption of the node.

This imperative to minimize energy consumption has had a profound impact on sensor node technology. To operate on ultra-low power, the microcontrollers constituting the computing platform of sensor nodes are severely constrained (cf. Figure 1.1). With their clock speeds in the order of a few megahertz and a mere few kilobytes of memory, they offer minimal computing power and storage space for sensornet applications and protocols. Likewise, the low-power radios providing the communication interface for sensor nodes are severely limited. Their theoretical maximal throughput is in the order of just tens to hundreds of kilobits per second; the effective throughput is even smaller, only a tiny fraction of these maximal values. Furthermore, the range of the radios is also small,
1.2. NETWORKING LOW-POWER WIRELESS NODES

from tens of meters indoors to hundreds of meters outdoors, which in typical applications means that special routing measures need to be taken to allow two arbitrarily chosen nodes to exchange information. Finally, the wireless internode connectivity offered by the radios exhibits a number of peculiar phenomena: it is highly irregular, even seemingly random, often asymmetric, and fluctuates over time depending on the environmental conditions. Such phenomena significantly impair the reliability of the internode communication.

These severe resource constraints and peculiarities of sensor nodes push the networking problem to an extreme. Compared to the networks forming today’s Internet, in which devices have ample computing power and storage, and links provide plenty of bandwidth and fairly reliable communication, sensornets are challenged in virtually all aspects. Therefore, to become the link in the evolution chain of the Internet which would bridge the digital and the physical worlds, sensornets have to deliver networking protocols which efficiently cope with all their challenges.

1.2. NETWORKING LOW-POWER WIRELESS NODES

Initially, when the sensornet research field was starting to form, it was argued that, while the experience accumulated during the development and the expansion of the Internet would be applicable to sensornets, the challenged nature and different requirements of sensornets justified reconsidering the overall architecture and organization of networking protocols [29]. In particular, it was argued that the severe resource constraints of sensor nodes, combined with large expected network sizes, might cause abandoning the layered [11] and end-to-end [110] design principles of the Internet. However, unlike the Internet, which had been designed to accommodate various applications, sensornets would be tailored to particular sensing applications, effectively being special-purpose networks. This in turn would allow researchers to overcome the challenges posed by sensornets by developing application-oriented networking protocols. Gradually, the most successful of those protocols would become foundations of generic Internet-like networking abstractions, suitable for standardization.

1.2.1. Decade of Research

Such a clean-slate approach has proved to be extremely fruitful, producing over a decade myriads of novel protocols related to virtually all aspects of networking. As those protocols were usually targeted at particular applications, they made
different, sometimes exotic, assumptions about the architecture and the objectives of their target system. Although such chaos precluded interoperability between protocols from different systems, it resulted in wide exploration of the protocol design assumptions, trade-offs, and solutions. In effect, as discussed in more detail in the next chapter, some important findings in low-power networking have been made. Moreover, since many of the protocols were implemented and evaluated in real-world deployments, it became possible to precisely identify the networking requirements of different sensornet application classes. As a consequence of those research activities, common, generic networking abstractions started to emerge from the initial protocol chaos.

As the sensornet field has been maturing, subsequent Internet-like networking abstractions have been identified. Currently, not only do they cover all layers of networking, but are also widely recognized and are becoming subject to standardization. At the lowest, physical communication layer, the standard most widely supported by hardware manufacturers is arguably IEEE 802.15.4 [65]. Similarly, 802.15.4, usually augmented with energy-saving mechanisms (e.g., low-power listening [104]), is used as the next communication layer, the link layer, which arbitrates node access to the wireless medium and duty-cycles node radios to conserve energy. In this way, protocols providing basic one-hop communication (i.e., communication between nodes within a radio range) are to a large extent standardized. As a result of the earlier research activities, there are also ongoing standardization efforts at subsequent communication layers, notably for protocols meshing nodes into a multi-hop network in a way that masks the unreliability and peculiarities of low-power wireless communication [9]. Finally, a recently demonstrated method for the Internet Protocol (IP) header compaction has enabled efficiently handling IP packets by the severely constrained sensor nodes [46]. This initiated another standardization process [96] which will ultimately allow each sensor node to be a fully-fledged Internet device.

By and large, all the research results and standardization efforts are compelling evidence that we are on the verge of realizing the aforementioned vision, in which thousands of wireless sensor nodes embedded in the surrounding environment bridge the digital and the physical worlds. There is, however, a component crucial to this vision which requires much more research attention: scalable point-to-point routing protocols for low-power wireless networks.

1.2.2. Networking Fabric

A routing protocol is responsible for finding paths in a network along which data are sent. In a typical network, a node can communicate directly only with a tiny
subset of all nodes, called the *neighbors* of this node. In a wired network, the neighbors of node A are those nodes which are connected to A with a network cable; in a sensornet, they are the nodes within the radio range of A. Therefore, to send a data packet to a non-neighbor node, a node first transmits the packet to one of its neighbors. The selected neighbor then forwards the packet to one of its own neighbors and so on, until the packet reaches the destination node. In other words, on the path from the *source node* to the *destination node*, the packet is forwarded by multiple *intermediate nodes*. The task of a routing protocol is selecting, at the source and each intermediate node, the neighbor node to which a packet will be forwarded next, the *next hop* on the path of the packet. This selection is typically based on the identities of the source and the destination nodes, their *routing addresses*. In short, without routing, the great majority of nodes in a network would not be able to communicate.

According to the architectural design of the Internet, point-to-point routing is the fabric binding all the Internet devices [83]. The layered design principle in the architecture of the Internet [11] identifies boundaries in the functionality of the network devices: the protocol layers. This principle effectively follows a philosophy of designing for heterogeneity, change, and uncertainty. The end-to-end principle [110], in turn, shifts as many of the layers as possible to the end-point devices. In this way, the functionality of the devices in the network core remains simple, which offers excellent scalability prospects. As a consequence of these two principles, the *only* common functionality of all, core and end-point, Internet nodes is point-to-point routing, in the form of the Internet Protocol (IP). The proliferation of the Internet is the best testimony that, with appropriate routing technology, such an architecture is extremely effective.

Yet, despite its fundamental role in the Internet, point-to-point routing was initially not deemed necessary for sensornets [29]. It was argued that, because of the constrained and application-oriented nature of sensornets, a sensor node might not need an identity, that is, a routing address [29]. Since the main application of sensornets envisioned at that time was environmental data collection, different routing primitives were considered. In particular, it was assumed that a sensor network would be accessed from the Internet through an application-level gateway. Such a gateway would inherently assume a role of a sink for the collected environmental data and a coordinator for the network. Therefore, tree-based all-to-one routing could be used for communicating environmental data from the sensors to the gateway and broadcast-based one-to-all routing for communicating control commands from the gateway to all the sensors. More elaborate applications could in contrast employ customized routing primitives, notably data-centric routing [49], in which the communication is for named data.
rather than node routing addresses. A number of compelling solutions, such as Directed Diffusion [45] and TAG [91], demonstrated how such routing primitives can incorporate localized algorithms and in-network data processing to facilitate building highly scalable sensornet systems.

Recently, however, point-to-point routing has been recognized as a fundamental networking component for sensornets [23, 46, 48, 133]. The argument that the resource constraints of sensor nodes preclude the Internet architecture, founded on point-to-point routing, is no longer defensible in face of the aforementioned research results and standardization efforts. Consequently, if every individual sensor is to be a fully-fledged member of the Internet, any Internet device must be able to route to that sensor and vice versa, as dictated by the architecture of the Internet. Similarly, the accumulated practical knowledge on how sensornets are used evidences that in many sensornet applications, especially those involving actuator nodes, any sensor node in a network should be able to route to any other node in the same network. In other words, all-to-one and one-to-all routing primitives are insufficient and, instead, explicit point-to-point routing support is required. Finally, even though it is still not clear whether localized algorithms, in-network processing, and data-centric routing will become prevalent in sensornets, they can all be implemented on top of point-to-point routing, as the example of the traditional Internet demonstrates.

Therefore, because point-to-point routing for sensornets is lacking sufficient research attention, this recent recognition that it is crucial networking functionality necessitates extensive in-depth studies. Even though due to the proliferation of the Internet point-to-point routing has been studied well, the challenging nature and different application requirements of sensornets force us to reconsider some of the earlier assumptions and to shift the research focus. Otherwise, without routing protocols suitable for sensornets, the progress in developing and adopting the sensornet technology may be impeded.

1.3. PROBLEM STATEMENT

This dissertation addresses the problem of scalable point-to-point routing for sensornets. The importance of the problem manifests itself not only in the fundamental role of point-to-point routing as the networking fabric, but also in the impact a point-to-point routing protocol has on protocols and applications on top. The effectiveness of a routing protocol directly impacts scalability, reliability, and efficiency of the higher-layer protocols and applications. If a routing protocol does not scale well, one cannot expect to be able to build applications that require
large networks. Likewise, if the reliability of a routing protocol is poor, it is
difficult to compensate for that in higher-layer protocols. Finally, if routing, the
basic communication primitive, is inefficient, networked systems built on top are
bound to be inefficient as well. For these reasons, the effectiveness of a routing
protocol is of the utmost importance.

Because of the challenging nature of sensornets and the requirements of their
applications, to be effective, a point-to-point routing protocol for sensornets has
to simultaneously ensure the following basic properties.

**Small Routing State:** Small routing state is essential to scalability and
efficiency. Since sensor nodes are severely constrained in terms of memory
and processing power, minimizing the state used for routing at each node
is crucial for such devices to form large networks. In addition, the smaller
the state of a protocol, the lower the control traffic to maintain this state, as
these two are usually correlated. This, in turn, is important for efficiency,
considering the bandwidth limitations of sensor nodes.

**Small Routing Stretch:** Routing stretch describes how costly the paths selected
by a routing protocol are, for example, what the ratio is between the number
of routing hops taken by a packet to the minimal possible number. Provided
that a one-hop transmission is reasonably reliable, the fewer hops there
are on a routing path, the lower the global resource consumption and the
end-to-end latency, and the higher the end-to-end message delivery rates.
Small routing stretch means that the cost of a route selected by a routing
protocol is close to the cost of the optimal route. Therefore, small routing
stretch is crucial for efficiency and reliability.

**Robustness:** Sensornets experience message loss and topology and connectivity
changes due to node failures, power outages, and interactions with the
surrounding environment. The larger the network, the more changes
are expected. Robustness to changes is thus important for scalability
and reliability. More specifically, to minimize resource consumption and
disruption of higher-layer protocols and applications, a routing protocol
must be resilient to message loss and must recover after a change in the
network with minimal traffic and latency.

In addition to these three basic performance-related properties, an important
functional feature of a routing protocol for sensornets is the ease of configuration
and management. On the one hand, the protocol should introduce some
configuration parameters that would allow for fine-tuning its performance for
particular applications and deployments. On the other hand, as many sensornet
applications necessitate large node populations, the deployment, configuration, and subsequent maintenance of such systems can be laborious. Therefore, the number of configuration parameters should be small and their impact on the protocol performance should be well understood. This implies that the protocol should be self-managed to a large extent. For example, it should not require human intervention to fill in the routing state of each node. Instead, when deployed, the nodes should autonomously synthesize their own routing state. Likewise, whenever the node population or network connectivity changes, the nodes should account for such changes autonomously.

All the above requirements lead to the following concise formulation of the problem this dissertation aims to address.

Given the importance of point-to-point routing for sensornets, the problem this dissertation aims to address is providing a point-to-point routing protocol for sensornets that ensures small routing state, small routing stretch, and robustness. In addition, the protocol should be self-managed to a large extent and should introduce only a small number of well understood configuration parameters for fine-tuning its performance.

The above problem formulation entails not only designing a protocol, but also implementing and evaluating it in the real world. As is demonstrated throughout the dissertation, implementation and real-world evaluation are crucial stages in the development of sensornet protocols.

1.4. THESIS

While point-to-point routing for sensornets has different assumptions and requirements than routing in the traditional Internet, the vast routing theory developed for the Internet can be, to a large extent, applied to sensornets, as discussed in detail in the next chapter. An important contribution of the routing theory is a discovery of a trade-off between routing state and routing stretch [70]. In particular, the theory explains what is possible when trading off state for stretch and identifies routing techniques which achieve a particular trade-off. Moreover, it studies the worst-case performance of different techniques and their expected performance in highly simplified, hypothetical network configurations.

One of those routing techniques, which is particularly attractive for sensornets, is hierarchical routing [66, 131]. In hierarchical routing, nodes are organized into a multi-level hierarchy of clusters such that the number of nodes
in the clusters at subsequent levels grows exponentially. The routing state of a node consists of the information about those clusters at each level that are within a certain “distance” from the node. In this way, the routing state of a node can be polylogarithmic with respect to the node population size, that is, even if the node population multiplies by a certain factor, the routing state of the nodes will grow by only a constant value. This means that the routing state maintained by the nodes can be very small. Moreover, in hierarchical routing, a change in the network topology has typically only local impact. This implies that hierarchical routing can be resilient to node failures and connectivity changes.

However, despite its merits, hierarchical routing has not received sufficient attention from the sensornet research community. More specifically, to the best of my knowledge, no hierarchical routing protocol has been implemented and evaluated experimentally. In contrast, as discussed in the next chapter, other techniques, which often require larger routing state or are less resilient, have recently been implemented and evaluated in the real world. Deprived of the corresponding information on practical aspects of hierarchical routing, policy makers in standardization committees and sensornet systems developers may be forced to make suboptimal choices regarding networking architecture for sensornets, in general, and point-to-point routing protocols, in particular. Just for this reason alone, hierarchical routing deserves more experimental research.

The deficit in research attention to hierarchical routing has two main origins [35, 94, 100]. Firstly, the routing theory evidences that, because of using very small state, hierarchical routing can have a large worst-case routing stretch [66, 94, 131]. This, in theory, can result in high global resource consumption and end-to-end latency, and low end-to-end message delivery rates, all of which are undesirable by sensornet applications. Secondly, building and maintaining a hierarchy of clusters required by hierarchical routing is a highly complex problem [35, 38, 42, 94]. This implies that developing efficient, robust, self-managed protocols in which nodes autonomously maintain the hierarchy is extremely challenging, especially for such peculiar, resource-constrained networks as sensornets.

This dissertation challenges the above view. Stated concisely, my thesis is as follows.

Hierarchical routing is a compelling point-to-point routing technique for large sensornets. In practice, not only does it offer small routing state, but also small routing stretch. Moreover, it is possible to provide robust, efficient, self-managed hierarchical routing protocols that work in the real world.
The rationale behind the thesis is a combination of a few observations and some past experience. First and foremost, the polylogarithmic routing state required by hierarchical routing is extremely attractive for large networks composed of highly resource-constrained sensor nodes. Moreover, although it is true that in some network topologies such a small state results in large stretch, the topologies of sensornets are typically “geometric,” that is, the average “distance” between nodes grows quickly with the node population size. The theory shows that, in such networks, the expected stretch of hierarchical routing can be very small [66, 70]. This means that, in practice, hierarchical routing for sensornets can potentially provide both: small state and small stretch. Finally, it is also true that constructing an optimal cluster hierarchy and maintaining it during the whole network lifetime may be extremely complex, even disregarding the resource constraints of sensor nodes. However, my past experience with scalable distributed systems, and in particular, with gossip-based protocols, indicates that, by relaxing some of the hierarchy properties, one can devise localized algorithms in which nodes self-organize into and autonomously maintain the hierarchy in a highly robust and efficient manner.

1.5. CONTRIBUTIONS AND OUTLINE

To evaluate the thesis, the subsequent chapters of this dissertation proceed through all stages of the protocol development process: from the analysis and design to the implementation and experimental evaluation. I believe that this holistic approach constitutes a sound methodology for verifying the validity of the thesis.

However, the outcome of this work stretches beyond confirming that hierarchical routing is a compelling point-to-point routing technique for large sensornets. More specifically, in addition to proving the thesis, the work demonstrates how different design choices within a hierarchical routing protocol affect its performance. It also show how hierarchical routing compares with other major representative routing techniques for sensornets. These contributions map to the chapters of this dissertation as follows.

Chapter 2 analyzes relevant background work. Firstly, it surveys the application domains of sensornets to illustrate how application requirements influence the hardware architecture of sensor nodes, and how these two together affect routing protocols. Secondly, it analyzes point-to-point routing theory and protocols developed for sensornets, discussing their merits, drawbacks,
and the trade-offs they involve. In particular, it demonstrates why hierarchical routing may be a compelling routing technique for sensornets.

Chapter 3 demonstrates how one can develop a hierarchical routing protocol which addresses the peculiarities of sensornets. More specifically, it discusses the theory behind and the design of PL-Gossip, a novel hierarchical routing protocol. The example of PL-Gossip illustrates how, by combining gossiping and localized probabilistic algorithms, nodes can autonomously build and maintain a hierarchical routing infrastructure in a robust and efficient manner. The solutions developed for PL-Gossip constitute the foundations of the work presented in the subsequent chapters. In short, Chapter 3 proves the second part of the thesis: it is possible to provide robust, self-managed, practical hierarchical routing protocols.

Chapter 4 evaluates hierarchical routing experimentally. To the best of my knowledge, this is the first such an evaluation for sensornets reported in the literature. To enable such an evaluation, a library for hierarchical routing has been designed and implemented. The library captures the common characteristics of PL-Gossip and other existing hierarchical routing protocols and, at the same time, identifies various design points where the protocols differ. These design points allow for experimenting with different design decisions and for exploring the design space. Using the library, an extensive experimental evaluation of hierarchical routing is conducted. The results prove the first part of the thesis: apart from small state, in sensornets hierarchical routing offers also small routing stretch. In addition, Chapter 4 constitutes a guide for hierarchical routing protocol designers over the available design space.

Chapter 5 compares hierarchical routing with other routing techniques. More specifically, it presents a point-to-point routing library for sensornets implemented specifically to conduct such comparisons. The library contains four competing representative techniques that, from the state-stretch trade-off perspective, cover the entire spectrum. The performance of the techniques is compared experimentally in terms of the state-stretch trade-off they offer. The results demonstrate that, compared to other techniques, hierarchical routing performs well in sensornets. For many applications, it arguably offers the most attractive state-stretch trade-off. Overall, not only does Chapter 5 reinforce the thesis, but it also demonstrates the relative performance of different routing techniques proposed for sensornets.
Chapter 6 concludes the dissertation. It summarizes the most important research findings. Furthermore, it identifies those aspects of the presented work, and of scalable point-to-point routing protocols in general, which have yet to be addressed in order to provide production-quality routing solutions. Some of these issues require additional research activities.

Moreover, the above research contributions involved a considerable amount of additional work. This additional work has been moved to the appendices. Appendix A contains the proofs of the lemmas from Chapter 3. Appendix B presents KonTest, a sensor network testbed which has been built at Vrije Universiteit Amsterdam to conduct the experiments presented in this dissertation. Appendix C gives an overview of the sources of the hierarchical routing library described in Chapter 4 and the sources of the general point-to-point routing library described in Chapter 5. Finally, Appendix D contains Dutch and Polish summaries of the dissertation.

ONE of the consequences of the proliferation of the Internet is an extensive volume of research on point-to-point routing. The fundamental role of routing in the Internet compelled researchers to develop models and metrics that allow for understanding the performance of routing protocols. Similarly, the continuous growth in the number of Internet devices inspired work on scalable routing protocols, which can handle sheer node populations with reasonable resource demands. By and large, point-to-point routing for the Internet is a well-studied problem.

However, while many of those results are applicable to sensornets, the challenging nature and different application requirements of sensornets force reconsidering some of the earlier assumptions and shifting the research focus. In particular, in contrast to Internet routers, which have ample computing power, storage, and bandwidth, the resources of sensor nodes are severely constrained. Likewise, while Internet routers utilize their vast resources to efficiently handle large volumes of data, sensornet applications typically involve low data rates, but require minimal energy consumption in return. In the same fashion, the
unreliable and hardly predictable low-power wireless communication of sensor nodes is completely different from the optical wired communication employed by the Internet routers. Such differences require sensornet protocols to emphasize different aspects of routing and to address a number of novel challenges.

This chapter analyzes the requirements of point-to-point routing protocols suitable for sensornets and discusses related work. To this end, Section 2.1 starts with examples of well-recognized application domains of sensornets. With the usage scenarios of sensornets being illustrated, Section 2.2 discusses how these scenarios impact the routing protocols, starting from the hardware architecture of sensor nodes to the solutions in the protocol stack. It also explains why the design goals for sensornet point-to-point routing protocols, set forth in Section 1.3, are unlikely to change in the foreseeable future. Finally, Section 2.3 surveys related work on routing protocols. It gives an overview of both theoretical and experimental results.

2.1. SENSORNET APPLICATIONS

The ability to deploy tiny wireless nodes equipped with sensors and actuators in the surrounding environment to relay data about the environment and to interact with that environment enables a number of novel applications. The requirements of these applications drive the hardware and software architecture of sensor nodes and thereby, the design of point-to-point routing protocols. To facilitate a better understanding of this relationship, let us briefly survey some sample, well-recognized sensornet applications.

2.1.1. Examples of Application Domains

Many of those surveyed applications have been deployed and evaluated in the real world; consequently, their requirements are known. Others are currently being developed or are still a future vision, but their requirements can be predicted to some extent. The common property of those applications is that they all can be supported by an Internet-like architecture for sensornets [47].

Scientific Exploration

Scientists from various Earth Sciences can make use of sensornets to study different aspects of the surrounding physical world in remote, uncontrolled, or hazardous places. Biologists can adopt sensornets to obtain better understanding of an ecosystem. For example, they can study the microclimate of a forest
canopy [130] or the habits of elusive animals [93] [123]. Geo-scientists can use sensor nodes to analyze geological features of the environment. For example, by gathering extensive data on permafrost [8] or rock strain [115], they can develop models which enable predicting disasters, such as avalanches and landslides.

The requirements that applications for scientific exploration set for an underlying routing protocol are understood to a large extent. The deployments to date have demonstrated that such applications usually require infrequent reporting of environmental data, such as temperature, pressure, or occupancy, to a central gateway [8, 93, 123, 130]. The gateway either assumes the role of a database or forwards the data to the Internet. In addition, asynchronous notifications may be reported by individual sensors or sensor groups for interesting events, for instance, rock strain exceeding a threshold or a burrow changing its occupancy [47]. From time to time, communication from the gateway to the nodes is required for configuration, retasking, and maintenance [93, 129]. Applications for scientific exploration can typically tolerate latencies and some loss of the collected sensor data [47] (provided that the data can be recovered later from the local flash memory of the nodes), which means that a routing protocol can trade off reliability for some other performance metrics.

### Structural Integrity and Seismic Activity Monitoring

Wireless vibration and deformation sensors can be used by civil engineers and seismologists. By analyzing data from such sensors, engineers can predict failures and schedule maintenance of factory machines [71], bridges [62], and buildings [140]. In particular, because of their visual unobtrusiveness, wireless sensor nodes are well-suited for monitoring the condition of historic monuments [14]. Likewise, seismologists can use wireless vibration sensors to monitor the seismic activity of volcanoes to predict potential eruptions [136] [137]. Remote monitoring with sensornets minimizes the risks associated with visiting volcanic sites, which are unavoidable when using traditional data-logging seismographs [136].

The traffic pattern of structural and seismic monitoring applications is similar to the traffic pattern of applications for scientific exploration in that it typically involves a centralized gateway. In addition, like in scientific exploration, latencies in structural and seismic monitoring can be tolerated to some extent. However, in structural monitoring applications, sensors are usually sampled at much higher frequencies and more data are reported to the gateway than in typical applications for scientific exploration [62] [136]. This is because a vibration signal contains high-frequency components. Moreover, vibration monitoring requires higher data delivery rates than scientific exploration [47]. The higher data quality is necessary
by algorithms for processing the vibration signal. To ensure such a quality, more communication from the gateway to the nodes may be required by transport protocols, for example, for acknowledging data receptions. Moreover, high best-effort packet delivery rates are desirable in the underlying routing protocol.

**Precision Agriculture and Industrial Automation**

Sensornets can also be applied in agriculture and industry. By deploying temperature, humidity, and light sensors in a field, farmers can obtain constant access to up-to-date fine-grained data on the current vegetation conditions [95]. Such data can be used to improve the efficiency of irrigation and pesticide administering — so-called *precision agriculture* — which increases crop yield and reduces costs [78, 95]. In industry, sensornets can be used to monitor production processes and machinery [47, 90]. Based on the data collected by the sensors along a production line, a human operator can precisely control the machines involved in the production process.

The traffic requirements of agricultural applications are similar to the requirements of applications for scientific exploration: low data rates and acceptable packet loss and latencies. The requirements of industrial monitoring applications vary, but many of them require a combination of infrequent, periodic, latency-tolerant data reporting and low-latency alarm notifications [47]. In addition, industrial applications generally require higher data delivery rates.

Like scientific exploration and structural monitoring, precision agriculture and industrial automation can be realized with a centralized gateway. Through the gateway, a human operator can receive reports and alarms from the sensors and, based on this information, can control the irrigation system or the production line. A closed-loop sense-and-control system, that is, a system without the human operator, can be implemented by programming the control logic into a central control computer connected to the gateway.

However, many people envision future closed-loop sense-and-control systems without a centralized component [1]. In such systems, sensor nodes would directly communicate with appropriate actuator nodes, thereby reducing resource consumption and latency.

**Home and Office Automation**

Another application of sensornet-based sense-and-control systems is home and office automation [103]. The goal of home and office automation is to minimize wiring in a building, notably, by eliminating the wiring involved in controlling the building environment. Sample applications of wireless sensor and actuator nodes
in building automation include fine-grained monitoring of energy consumption of building appliances [58], wireless control of light switches [117], automated control of heating, ventilation, and air conditioning (HVAC) systems [47], and automated security systems [99]. A future vision of building automation involves appliances equipped with sensors and actuators which communicate with each other to autonomously ensure certain desired properties of the building environment, including the room climate, the light intensity, or the aromas, to name a few [103]. This futuristic vision is often referred to as smart buildings.

Like in many of the above application domains, in building automation the traffic consists of infrequent, periodic reporting of data and asynchronous event notifications. The notifications should be reliable and should be delivered at human-scale latencies. For example, pressing a wireless switch controlling a light should guarantee that within a short period, measured in a human time scale, the light will turn on. Moreover, the communication in building automation can be decentralized as the sensors and actuators in appliances can communicate with each other directly rather than through a central controller. This approach avoids wasting resources and incurring additional latencies.

**Asset Tracking**

While in the above applications sensor nodes are assumed to be static, their untethered nature and small form-factor allow them to be mobile if necessary. This capability, in particular, makes sensornets suitable for tracking the location of mobile assets and sensing information about the assets themselves [75] [114]. For example, wireless sensor nodes can be used in logistics to locate containers and to provide some information about their content [47]. In hospitals, sensor nodes can track the location of medical equipment or staff, or even the location and the vital signs of individual patients [69].

Such applications, apart from mobile nodes, typically require a number of stationary ones, which have a well-known physical location and form a communication backbone. The mobile nodes usually infer their location based on their radio connectivity with the stationary nodes. They also use the backbone formed by the stationary nodes to report their location and sensor samples to data sinks. Since mobile nodes may lose connectivity with the backbone, they may be required to temporarily log their sensor samples locally during disconnection periods and to offload the logs when they regain connectivity — so-called delay-tolerant networking. All in all, the backbone formed by the stationary nodes must be prepared to communicate location reports, sensor samples, commands, and event notifications between any two of its nodes. A
stationary node should also be prepared to hand a mobile node off to another stationary node when the mobile node changes its location \[76\].

**Toward Ubiquitous Sensing**

Such a model, with stationary nodes forming the communication backbone and mobile nodes with intermittent connectivity with the backbone, can be adopted by metropolitan-scale sensor networks. Such networks are envisioned to perform numerous duties. For example, wireless sensors embedded in a road can monitor the road traffic and surface conditions, and can relay their observations through the backbone to traffic light controllers \[68\]. The controllers can then appropriately manage the traffic lights to improve the road throughput. They can also inform mobile in-car computers about traffic jams and road hazards. Metropolitan sensornets can monitor water pipes and sewer systems, the air quality on main avenues and at airports, can help in garbage collection and processing, or can assist in street parking. Their enormous potential stems from the fact that they can intertwine the digital world of the Internet with the surrounding physical world, thereby realizing the aforementioned visions of ubiquitous computing and the Internet of things.

**2.1.2. Common Features**

From the above examples, one can attempt to identify the common features of sensornet applications. This may be an unmanageable task because, although many sensornet application domains are similar in a number of aspects, their details do vary. There are, however, at least three common basic features which distinguish sensornets from traditional networks \[47\].

**Embedding in physical environment:** In traditional networks, such as the Internet, the physical placement of a node is typically determined by the access to power and network connectivity and the convenience of the users or the owners. In contrast, the applications of sensornets involve specific placement of nodes: near the points where sensing or actuation is to take place. Such embedding in the physical space has profound consequences. First and foremost, a sensor node has to be able to operate untethered, being powered by an autonomous power source, and to communicate wirelessly, for example, using a radio. The capability of wireless operation is necessary to embed sensor nodes in desired locations and to support node mobility. Furthermore, the physical location of a node often limits the possible size of the node; the same is true for mobility. As mentioned
2.1. Sensornet Applications

in the previous chapter, the progress in the miniaturization of power sources has a different pace than in the miniaturization of electronic circuits, and consequently, the autonomous power source is often the largest element of a node. This means that the size constraints of a node translate directly to the size constraints of the power source, which effectively limit the energy budget of the node, be it energy stored in a battery or energy harvested through solar panels, vibration, or other methods.

Large numbers of nodes in an administrative domain: The goal of sensornets — to enable sensing and acting upon features of the surrounding environment at an unprecedented scale and fidelity — entails possibly large networks. To ensure sufficient granularity of sensing and to cover all actuation points, the node density may be high. Therefore, to monitor a phenomenon spanning a certain physical area or to control various objects in the area, a large node population may be necessary, depending on the size of the area and the density of the sensing and actuation points. A single application deployment may thus involve hundreds or even thousands of nodes, which will likely operate under a single administrative domain. While large node populations in an administrative domain are common in some wired networks, such as in data centers, they are unusual for most other wireless networks. In WiFi networks, for instance, the administrative domain of a single user typically consists of only her wireless-enabled laptop. Likewise, in cellular networks, a single user is usually responsible for one or at most a few mobile phones.

Low total cost of ownership per node: The large possible number of nodes in a single administrative domain, implies that the per-node cost of producing, deploying, and maintaining a sensornet has to be reasonable. Wireless operation already reduces the deployment cost by eliminating the costs of cables and the effort involved in installing them. To minimize the maintenance cost, nodes have to operate unattended for extended periods of time on modest power sources. This implies that they have to consume minimal amounts of energy. In addition, they should be easily configurable and manageable. Finally, to avoid excessive production costs, the cost of components constituting a node has to be reasonable as well.
2.2. CONSEQUENCES FOR ROUTING

Starting from the hardware architecture of sensor nodes to the solutions in the protocol stack, the above application requirements and features of sensornets fundamentally affect all aspects of routing and, in general, networking. As a consequence, the common claim that sensornets constitute a new class of computing is by no means exaggerated.

2.2.1. Hardware and Physical Limitations

A wireless sensor node can be thought of as a fully-functional, albeit peculiar, tiny networked computer. It usually consists of a microcontroller (MCU), a radio with an omnidirectional antenna, interfaces with input-output components, and an autonomous power source (see Figure 2.1). Let us focus on the MCU and the radio, as their constraints directly impact the design of routing protocols. The power source is typically a set of batteries or some energy harvesting system, such as a solar panel. The input-output components may include low-power sensors and actuators, light emitting diodes, and external flash memory, to name a few.

![Figure 2.1: An overview of hardware components constituting a sensor node. A node typically consists of: a microcontroller (MCU), a radio with an omnidirectional antenna, interfaces with input-output components, such as flash memory, sensors, and actuators, and an autonomous power source, such as a battery or some energy harvesting system.](image)

The MCU and the radio constitute respectively the computing and the communication platform of a node. In general, the MCUs and radios employed by most sensornet platforms are relatively simple and provide fewer resources than those in other computing classes. In return, they offer small form-factor
2.2. CONSEQUENCES FOR ROUTING

### Table 2.1: Representative examples of MCUs suitable for sensornets.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel</td>
<td>ATmega128L</td>
<td>2002</td>
<td>8-bit</td>
<td>2.7–5.5</td>
<td>4</td>
<td>128</td>
<td>0.95</td>
<td>5</td>
<td>6</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>ATmega2561</td>
<td>2005</td>
<td>8-bit</td>
<td>1.8–5.5</td>
<td>8</td>
<td>256</td>
<td>0.9</td>
<td>1</td>
<td>6</td>
<td>81</td>
</tr>
<tr>
<td>Freescale</td>
<td>HCS08</td>
<td>1993</td>
<td>8-bit</td>
<td>4.5–5.5</td>
<td>1</td>
<td>32</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>HC08</td>
<td>2003</td>
<td>8-bit</td>
<td>2.7–5.5</td>
<td>4</td>
<td>60</td>
<td>7.4</td>
<td>1</td>
<td>10</td>
<td>144</td>
</tr>
<tr>
<td></td>
<td>MC13213</td>
<td>2007</td>
<td>8-bit</td>
<td>2.0–3.4</td>
<td>4</td>
<td>60</td>
<td>6.5</td>
<td>35</td>
<td>10</td>
<td>81</td>
</tr>
<tr>
<td>Jennic</td>
<td>JN5121</td>
<td>2005</td>
<td>32-bit</td>
<td>2.2–3.6</td>
<td>96</td>
<td>64</td>
<td>4.2</td>
<td>5</td>
<td>&gt;2500</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>JN5139</td>
<td>2007</td>
<td>32-bit</td>
<td>2.2–3.6</td>
<td>96</td>
<td>192</td>
<td>3.0</td>
<td>3.3</td>
<td>&gt;2500</td>
<td>64</td>
</tr>
<tr>
<td>TI</td>
<td>MSP430F149</td>
<td>2000</td>
<td>16-bit</td>
<td>1.8–3.6</td>
<td>2</td>
<td>60</td>
<td>0.42</td>
<td>1.6</td>
<td>6</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>MSP430F1611</td>
<td>2004</td>
<td>16-bit</td>
<td>1.8–3.6</td>
<td>10</td>
<td>48</td>
<td>0.5</td>
<td>2.6</td>
<td>6</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>MSP430F2618</td>
<td>2007</td>
<td>16-bit</td>
<td>1.8–3.6</td>
<td>8</td>
<td>116</td>
<td>0.5</td>
<td>1.1</td>
<td>1</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>MSP430F5437</td>
<td>2008</td>
<td>16-bit</td>
<td>1.8–3.6</td>
<td>16</td>
<td>256</td>
<td>0.28</td>
<td>1.7</td>
<td>5</td>
<td>196</td>
</tr>
<tr>
<td></td>
<td>CC2430</td>
<td>2007</td>
<td>8-bit</td>
<td>2.0–3.6</td>
<td>8(4)∗</td>
<td>128</td>
<td>5.1</td>
<td>0.5</td>
<td>4</td>
<td>49</td>
</tr>
</tbody>
</table>

Key parameters include the supply voltage (VCC), the current consumption in the active mode (at 3 V and 1 MHz if possible) and the sleep mode, the wakeup time from the sleep to the active mode, the amount of program memory (ROM) and random access memory (RAM), and the required circuit board area. Note (*) that CC2430 has 8 KB of RAM, but only 4 KB are retained when the MCU enters the sleep mode. The table was compiled based on a paper by Dutta et al. [26] and manufacturer datasheets. Although all the effort was put to ensure that the data are correct, different manufacturers provide different metrics and, moreover, their data differs slightly from the empirical data reported in various papers. Therefore, the table should be treated as an overview of current trends rather than the definitive source of information on the performance of particular MCUs.

and low price, and strive to ensure minimal energy consumption, as these are the properties that are most desired by sensornet applications.

### Microcontroller Constraints

An MCU of a sensor node integrates the processing unit, random access memory (RAM), program memory, and other peripherals, such as bus controllers or analog-to-digital converters. Integrating all these components into a single chip reduces the production costs and form-factor. For this reason, MCUs which also integrate low-power radios have started to appear recently. Examples of MCUs suitable for and employed in today’s sensor node platforms are listed in Table 2.1.

Apart from integrating several components, the architectures of the MCUs used in sensornets have a common major objective: minimizing energy consumption. To this end, they aim at simplicity: a minimal instruction set, low processor frequencies, small amounts of memory, etc. Such a simple design involves fewer transistors and, thereby, results in lower energy consumption when an MCU is active.

Moreover, to facilitate saving energy when an MCU is idle, the MCU
architectures feature several operation modes, in particular, at least one ultra-low-power sleep mode. An application switches the MCU into a sleep mode when there is no work, and activates it back when some work needs to be done [67]. Likewise, peripherals, such as bus controllers and analog-to-digital converters, are activated only when they are necessary [67]. The simplicity of an MCU thus also ensures that switching between different operation modes is as fast as possible.

Considering that in a sleep mode the power draw of an MCU is a few orders of magnitude lower than in a normal active mode (cf. Table 2.1), entering the sleep mode is the most effective way for the MCU to save energy. Since sensornet applications typically involve low data rates, their MCU utilization is often just a few percent. Similarly, they use the MCU peripherals relatively infrequently. Consequently, the MCU typically remains in the sleep mode for the great majority of time [67].

Therefore, because sleep mode is the dominating operation mode of an MCU, the MCU architectures for sensornets have to minimize the power draw in this mode. Since in sleep mode most energy consumption is attributed to preserving the contents of random access memory (RAM), the majority of today’s MCUs limit their RAM to 10 KB (cf. Table 2.1); otherwise, their sleep power and wakeup time would be several orders of magnitude larger (e.g., Jennic JN5121 in Table 2.1). To further reduce the sleep power, some MCUs support selective RAM retention, that is, only selected portions of RAM are preserved when an MCU enters sleep mode while the rest is discarded (e.g., TI CC2430 in Table 2.1). This feature is not yet common, though, and it is still unclear whether it will be desirable.

The severe limitations in the amount of RAM constitute the major challenge MCUs pose for sensornet protocols. Protocols at all layers of the network stack use RAM to maintain some state. For example, a routing protocol typically maintains at each node the information on the routes from that node to a number of other nodes. This state often depends on the total number of nodes in the network. However, severely constrained RAM precludes large state. Even though it may be possible for a protocol to store its state in nonvolatile external storage, it is not necessarily a good solution. Protocol state typically changes and has to be refreshed, for instance, link quality estimates and the resulting next-hop neighbor candidates may change quite frequently. However, writing to nonvolatile storage, such as flash or EEPROM, has a high energy cost and involves significant latency. In addition, flash and EEPROM have a limited number of write cycles. For these reasons, sensornet protocols typically maintain their state in RAM.

Therefore, to support networks of hundreds or even thousands of nodes with
just a few KB of RAM per node, sensornet protocols have to be scalable. More specifically, the state they maintain should ideally not depend on the total number of nodes or, more practically, should grow gracefully with the node population size. A gracefully growing state is sublinear with respect to the number of nodes, for example, it can be a polylogarithmic function of the total number of nodes.

Other MCU constraints have a smaller impact. In particular, while the program memory is also limited, 128 KB has been more than sufficient for the applications to date. There are applications that require around 60 KB of program memory, so some MCUs, such as the popular TI MSP430F1611, may be too constrained. Nevertheless, since program memory does not contribute to the sleep power draw, a particular limit is mainly a matter of the MCU cost and of architectural constraints rather than an energy-related issue. Likewise, due to low typical MCU utilization, the computing power provided by today’s MCU is usually sufficient. Again, there are scenarios in which the processing speed is too low, for example, a high-speed data transfer over a serial cable from a computer to an MSP430F1611-based sensor node. All in all, however, the major challenge posed by MCUs for sensornet routing protocols is the severely limited RAM.

Radio Constraints

Like MCUs, to minimize energy consumption, production costs, and form-factor, the radios employed in sensornets aim at simplicity. There are numerous such low-power radios available, not only for sensor nodes, but also for other wireless devices, including key-less car door openers and wireless game controllers. The initial sensornet platforms adopted narrowband radios, which are not as robust to narrowband interference as wideband radios, but are less complex, draw less power, and are less expensive. Currently, however, mainly due to the introduction of IEEE 802.15.4, wideband radios have become common. Sample low-power radios suitable for and used in sensornets are presented in Table 2.2.

A low-power radio, be it narrowband or wideband, has a significantly constrained transmit power, often to around zero dBm. This limitation has two objectives: first, to make the transmit current low, and second, to prevent wasting energy on long-range transmissions. The motivation for the first objective is reducing the power draw during transmissions. The rationale behind the second one, in turn, is the fact that the energy requirements increase polynomially with range and thus, it is more efficient to perform multiple short-range transmissions than one that is long-range.

However, because of low transmit power, two sensor nodes with line-of-sight visibility may not be in each other’s transmission range and, consequently, may
Table 2.2: Representative examples of radios suitable for sensornets. Key parameters include the supply voltage (VCC), the current consumption during transmitting with a given output transmission power (TX/Power), the current consumption during receiving and listening (RX), the current consumption in the sleep mode (Sleep), the wakeup time from the sleep to the active mode, the throughput, and the required circuit board area. The table was compiled based on a paper by Dutta et al. [26] and manufacturer datasheets.

For the same reasons as Table 2.1, this table should be treated as an overview of current trends rather than the definitive source of information on the performance of particular radio chips.

not be able to communicate directly. Moreover, limited transmit power makes penetrating obstacles more difficult, and therefore, two nodes close to each other may not be able to communicate due to an obstacle in between. As a result, a packet from one node to another node may need to be retransmitted by multiple intermediate nodes to compensate for the low radio range and to overcome communication obstacles. In other words, sensornets require multihop routing protocols in which essentially every node can act as a router.

Lower transmit power also reduces the signal-to-noise ratio, thereby making the communication more susceptible to interference from the surrounding environment and nearby electronic devices, like microwave ovens. Moreover, it also makes other wireless devices, such as laptops or WiFi-enabled handhelds, less likely to identify an ongoing transmission and hence, less likely to delay their own transmissions to avoid causing interference. In effect, because of low transmit power, sensornet radios exhibit more packet loss than other higher-power wireless radios, such as WiFi.

In general, due to radio signal fading, interference, noise, hardware variations, multipath, attenuation, and other effects, the packet reception rate (PRR) of a low-power radio varies greatly between virtual wireless internode links provided by the radio. Although there exist links which offer nearly 100% PRR, a large fraction of links falls in a so-called transitional region, that is, the PRR of such
2.2. CONSEQUENCES FOR ROUTING

Links vary anywhere between 0% and 100% \[17, 138, 143\]. Moreover, many links are asymmetric, that is, the PRR when node A transmits to node B is different than the PRR when node B transmits to node A \[15, 92\]. Asymmetric links are problematic when acknowledgment-based mechanisms are used to improve packet delivery, because they can lead to unnecessary retransmissions in a situation in which packets from A to B are delivered, but the acknowledgments from B to A are lost. Finally, the PRR of a single link can vary in time, depending on the noise and the impact of the surrounding environment \[82, 118, 119\].

Routing protocols for sensornets have to efficiently cope with all these peculiarities of wireless links. First and foremost, they have to select routes that involve few intermediate hops, as this minimizes the number of places in which a routed packet can be lost. Equally important is also minimizing the number of transmissions necessary to successfully deliver a packet over a route. To this end, the protocols have to continuously measure the quality of the links, for example, in terms of PRR, and reject links that are asymmetric or are of low quality \[21, 34, 138\]. The quality measurement is performed based on the traffic and physical channel quality indicators, which are often provided by radios. In this way, a routing protocol can ensure that its routes consist of high-quality links, which makes it more likely that fewer transmissions will be necessary at each routing hop. Finally, routing protocols have to recover from temporal packet loss on a hop-by-hop basis, in addition to the end-to-end recovery adopted for the Internet. For example, by acknowledging packet reception at each intermediate node, the node can detect whether a packet has been lost and can decide to retransmit it.

Apart from the above limitations stemming from the low transmit power, low-power radios are also constrained in terms of throughput. The maximal theoretical throughput of most sensornet radios is below 500 kilobits per second, but for some, this value is even around 20 kilobits per second. The low throughput of sensornet radios, as compared to what can be achieved with higher-power radios, is a result of the power constraints and relative simplicity of the radios. In addition, even if the radios could support higher data rates, because of the limited RAM and computing power, the MCUs would likely not be able to buffer and consume the received data.

Worse yet, the effective throughput of low-power radios is just a small fraction of the maximal values. For many low-power radios, the current drawn when receiving or just idly listening for packets is similar to the current drawn when transmitting; for some, the receive/listen current is even higher (cf. Table 2.2). Moreover, when active, the radio typically consumes significantly more energy than any other component of a sensor node (compare, for instance, Table 2.1 with
Table 2.2). Therefore, to minimize energy consumption, like MCUs, low-power radios have to be inactive as much as possible, to avoid wasting energy on idle listening, and should be activated only to transmit and receive packets — so-called duty cycling [13, 77, 79, 104, 132, 141, 142]. To stay within their energy budget, most sensornet applications require a duty cycle of just a few percent, that is, the aggregate duration of all periods of radio activity cannot exceed a few percent of the total targeted network lifetime. For example, to enable a lifetime of approximately three years on a pair of AA batteries, the duty cycle of the CC2420 radio in a popular TelosB sensor node has to be around 1% [106]. However, a duty cycle of a few percent implies that the effective radio throughput is also a few percent of the already low maximal throughput.

Low radio throughput limits the amount of information nodes can communicate and increases communication latency. This has two major consequences for routing protocols. Firstly, it reinforces the requirement of small, gracefully scaling state. Since the protocols communicate not only to route packets, but also to maintain their state, and since the volume of such maintenance traffic is typically correlated with the amount of state, the throughput limitation requires the maintenance traffic, and hence, the state, to be low and scalable with the number of nodes. Secondly, the increased latency resulting from the throughput limitation necessitates routing protocols which choose short routing paths, in terms of both the number of hops and the expected number of transmissions. By choosing such paths, the protocols minimize the aggregate end-to-end routing latency and the aggregate global amount of resources consumed for routing, including energy, packet buffer space, and bandwidth.

To sum up, the relatively short range, low throughput, lossy nature, and other peculiarities of wireless low-power links are the major challenges low-power radios pose for routing protocols.

2.2.2. Deployment Constraints

Apart from the challenges stemming from the severe hardware constraints, the way sensornets are deployed introduces additional problems for routing protocols. The physical placement of nodes is typically dictated by the desired location of sensing and actuation points, rather than the availability of power, connectivity, and users’ convenience, as in traditional networks. One of the consequences of this constraint is that in most sensornet applications it becomes difficult to plan and guarantee particular network connectivity. Even if nodes are static, their placement may require them to communicate through physical
2.2. CONSEQUENCES FOR ROUTING

objects, which have various effects on radio communication, or may make
them subject to interference and noise from other nodes, higher-power wireless
networks, electronic devices, or the surrounding environment. Although due
to a higher signal-to-noise ratio nearby nodes usually provide high-quality
links, the unpredictable nature of low-power wireless connectivity may result in
higher-quality links for nodes further apart. As a result, it is virtually impossible
to engineer the network topology. In particular, every node may be required to
route messages for other nodes.

Moreover, another consequence of the embedding in the physical space is that
changes in the network topology are common. Since the quality of low-power
wireless links varies in time, depending on the impact of the environment, the
connectivity between nodes will change as well. Furthermore, some nodes
will fail: they may run out of energy, be physically destroyed, or simply stop
functioning, for example, due to water penetrating their casings. The exposure of
nodes to the surrounding environment makes the latter types of failures more
likely to happen in sensornets than in traditional networks. Moreover, as a
sensornet may be deployed in a remote, uncontrolled, or hazardous environment,
physical access to nodes may be limited. Consequently, manual recovery
after failures becomes problematic. In other words, the node population and
connectivity may change in a way that is difficult to control; the larger the
network, the more such changes will occur.

For these reasons, a routing protocol for sensornets should be robust. It should
be resilient to changes in the node population and connectivity: the impact of a
node failure or a link change should preferably have only a local scope, so that
nodes further away could normally route between each other. Moreover, nodes
running the protocol should be able to autonomously recover after a failure. Such
recovery should be fast to minimize the disruption of applications and should
consume minimal amounts of resources. In particular, the protocol should be
able to autonomously find alternative routes, if they exist, to prevent permanent
disconnection of some nodes from each other.

In general, as configuration of a large network is challenging, routing
protocols should be self-managed to a large extent. A node running a routing
protocol should be able to autonomously discover its neighbors, measure the
quality of the links to those neighbors, fill in the state necessary for routing, and,
possibly, also obtain a routing address. It should also be able to maintain this
state autonomously, so as to account for any changes in the network topology.
If necessary, the protocol should expose few configuration parameters, which
should only be used for fine-tuning its performance, and the impact of which
should be well-understood.
2.2.3. Putting Everything Together

Like similar analyses to date [35, 94], the reader’s own analysis of all the hardware and deployment constraints of sensornets would lead to the three performance-related properties which a sensornet routing protocol has to ensure, and which were set forth in Section 1.3. Let us briefly revisit these properties and discuss why they are unlikely to change in the foreseeable future.

Summary of Protocol Requirements

First, a routing protocol for sensornets has to provide small routing state. Small state designated for routing purposes is crucial because of the hardware constraints of sensor nodes. The MCUs adopted for sensor nodes have severely limited RAM. Consequently, whatever state a routing protocol maintains has to fit in this RAM, even if the network is large. Moreover, the effective throughput of the radios used by sensor nodes is also very small, and thus, the volume of traffic to maintain this state has to be small as well. Since the volume of maintenance traffic is typically correlated with the size of the state, the smaller the state, the smaller the traffic.

Second, the routing paths a routing protocol selects have to be close to the optimal paths. Nearly optimal paths minimize global resource consumption in the network: the fewer hops and transmissions are necessary to deliver a packet, the less buffer space, bandwidth, and energy is consumed by routing the packet. Moreover, nearly optimal paths improve the best-effort end-to-end packet delivery rates: the fewer the hops on a routing path of a packet and the better the links between such hops, the fewer the places in which the packet can be lost and the lower the probability of a loss. The quality of paths is usually measured using routing stretch, which is often decomposed into the following two metrics: hop stretch, the ratio of the number of routing hops on a routing path to the number of hops on the shortest possible path, and transmission stretch, the ratio of the number of transmissions on a routing path to the number of hops on this path. It is thus crucial for a routing protocol to offer small routing stretch.

Third, a routing protocol has to be robust. The embedding of sensor nodes in the physical environment exposes them to the interactions with the environment. Such interactions result in the changes in the node population and the internode connectivity, which disrupt protocols and applications. For instance, a node can fail or run out of power, or a communication obstacle between some nodes may emerge or disappear. Robustness of a routing protocol is thus crucial to minimize such disruptions, that is, they should have a limited impact and a routing protocol should be able to recover after them with reasonable traffic and latency.
2.2. CONSEQUENCES FOR ROUTING

Figure 2.2: Technological divergence of sensor nodes from Internet routers. The figure depicts evolving computer classes based on technology and design goals, which illustrates how the technological progress for Internet routers differs from the one for sensornets. The key design goal for Internet routers is performance, which necessitates resources, notably computing power and memory; price, energy consumption, and form-factor are of secondary importance. For sensornets, in contrast, the crucial metrics are energy consumption, price, and form-factor; the amount of resources available is determined by what is possible given some target values of these three metrics. The figure was adopted from the paper on Bell’s Law [7].

Apart from these performance-related properties, a routing protocol should also ensure a few functional properties, notably, it should be self-managed to a large extent. Nodes running the protocol should be able to autonomously deal with the peculiarities of sensornets and the influence of the environment. The protocol should introduce only a minimal number of well-understood parameters, which could be used for fine-tuning its performance. Self-management capabilities are necessary to minimize the setup and maintenance costs involved in deploying a large sensornet and keeping it operational.

Future Perspective

The above requirements for a point-to-point routing protocol for sensornets are unlikely to change in the foreseeable future. The application examples and the resulting hardware and deployment constraints illustrate that sensornets are on the other side of the technology spectrum than today’s Internet routers. The main objective of the Internet routers is to efficiently handle the continually increasing
traffic; the costs and energy consumption are not as important. To this end, the Internet routers are dedicated, high-end devices with customized memory and processing units [109]. As a side result, however, they are expensive and consume lots of energy. Moreover, the optical high-speed wired communication employed in the Internet core is more reliable and predictable than the low-power wireless communication in sensornets. In particular, packet loss in the Internet usually occurs due to congestion, which can be controlled to some extent, rather than due to some external unpredictable environmental impact. Therefore, compared to the Internet routers, sensor nodes are challenged in all aspects. Not only does a resource-constrained node have to act as a router, but it also has to efficiently cope with all the peculiarities of wireless low-power communication and the consequences of embedding in physical space.

This discrepancy between the Internet routers and sensor nodes is unlikely to diminish in the foreseeable future; conversely, it is likely to grow. Because of different objectives, technological advances in the Internet routers and sensor nodes will aim at different performance metrics, as illustrated by Figure 2.2. Novel bandwidth-hungry applications, such as high-definition video on demand and live television streaming, will demand more and more resources from the Internet routers. In contrast, to start being adopted, many envisioned sensornet applications require more energy-efficient, less expensive, and smaller sensor nodes, so that they can be deployed in large numbers, ranging from hundreds to thousands, for periods of time of at least several years to preferably a few decades or even more.

For these reasons, the networking challenges stemming from the constraints of sensornets are likely to still be valid in the foreseeable future. Therefore, with the technology progress enabling larger and longer lived networks, the research results on small-state, small-stretch, robust, self-managed routing protocols for sensornets are likely to gain even more importance.

### 2.3. RELATED WORK ON ROUTING

However, developing small-state, small-stretch, robust, self-managed point-to-point routing protocols is not trivial. Apart from meeting the standard robustness requirement, which in sensornets is particularly challenging, ensuring simultaneously small state and small stretch is a major problem. The theory of point-to-point routing identifies and analyzes a trade-off between routing state and stretch [70], which implies that the smaller the state the larger the stretch, and vice versa. There exists a whole spectrum of routing techniques which trade
off state for stretch at different granularity.

2.3.1. Shortest-Path Routing

One end of the state-stretch spectrum of routing techniques corresponds to shortest-path routing. The objective of shortest-path routing is to allow each node to route to any other node along the shortest possible path, which guarantees the minimal routing stretch; the routing state maintained by nodes is of secondary importance. In other words, shortest-path routing is aimed at environments in which storage space and bandwidth for the control traffic are plentiful, but which require efficiently handling the actual routed traffic. For instance, shortest-path routing is the prevalent technique for intra-domain routing in the Internet [70].

In shortest-path routing, the routing address of a node is a flat unique identifier assigned to the node. The routing state of a node, in turn, involves a routing table, conceptually, with one routing entry for every other node in the network. A routing entry for a node contains, among others, the unique identifier of that node, the cost of the shortest route to that node, and the link-layer address of the next-hop neighbor on that route. When routing a packet, a node looks up in its routing table an entry with the unique identifier of the destination node. From the entry, it extracts the link-layer address of the neighbor on the shortest route to the destination. It then forwards the packet to that neighbor, which repeats the process, and so on, until the packet reaches the destination. Since each forwarding node chooses as the next hop a neighbor that is on the shortest route from the node to the destination, overall, from the source to the destination, the packet travels along the shortest possible route.

To perform shortest-path routing nodes have to build and maintain their routing tables. This can be done in a variety of ways. In wired networks, for example, route maintenance is typically performed by a link-state protocol, such as OSPF [98], a distance-vector protocol, such as RIP [44], or a hybrid of the two, such as LVA [6]. In link-state protocols, each node periodically floods its neighbor connectivity information to all other nodes. By also gathering such information, the node builds its own connectivity map of the whole network. Based on this map, using Dijkstra’s algorithm [18] p. 595, it computes the shortest paths to all other nodes and stores them in its routing table. In distance-vector protocols, in turn, instead of flooding connectivity information throughout the whole network, each node periodically exchanges its routing table with its neighbors. Based on this information, using the Bellman-Ford algorithm [18] p. 588, it gradually calculates or updates routes to all other nodes in the network. Distance-vector protocols have lower message complexity of the control traffic, but they react...
more slowly to changes in the network as compared to link-state protocols.

In an alternative route maintenance technique, designed particularly for mobile ad hoc wireless networks, nodes do not continuously exchange route information. Instead, a route between two nodes is created on demand when the nodes need to communicate. For example, in protocols such as AODV [102] and DSR [60], when a source node needs to communicate with a destination node for which it does not have a routing entry, it floods a route request packet throughout the network. While spreading through the network, the route request effectively sets up a spanning tree rooted at the source node, that is, all nodes can create a routing entry with the shortest path to the source node. As soon as the request packet reaches the destination node, the destination uses such an entry to route a reply packet back to the source. In this way, the source and all intermediate nodes forwarding the route reply packet can also create a routing entry for the destination, which enables bidirectional routing.

To sum up, shortest-path routing provides the minimal possible routing stretch — equal to 1. To this end, however, it requires a routing state which is linear with respect to the node population size. As is shown in Chapter 5, a linear routing state in sensornets precludes scalability beyond a few tens of nodes. Although there are tricks to improve the scalability of shortest-path routing, their effects and applicability to sensornets are limited. For example, CIDR mechanisms used to compact the routing table sizes [37] are possible in the Internet because the connectivity in wired networks can be engineered. In contrast, as discussed in Section 2.2.2, the connectivity in sensornets is unpredictable, and thus, employing CIDR would be challenging. Likewise, protocols which fill the node routing tables on demand may be applicable to some sensornet deployments [36], especially those in which few pairs of nodes communicate. In many applications, however, an increase in the node population entails an increase in the number of communicating node pairs, which may again preclude scalability beyond a certain point. All in all, irrespective of the tricks used to reduce the node state, a linear bound on the state is an inherent drawback of shortest-path routing, and, as such, it will always limit scalability.

2.3.2. Constant-State Routing

Considering their memory and bandwidth limitations, sensornets require protocols with asymptotically smaller routing state. Ideally, the routing state should be independent of the node population size. Such constant-state routing techniques constitute the other end of the state-stretch trade-off spectrum.
Centralized Routing

One approach to providing constant-state routing is centralized routing, which has been adopted by protocols such as CentRoute [121] and the proposed IP-based network architecture for sensornets [46, 47]. Centralized routing makes use of the systems aspects of many sensornet applications, notably the common deployment scenarios and traffic patterns. In many applications, the dominating traffic pattern is many-to-one data collection in which sensor nodes report their data to a gateway. Therefore, one can optimize routing for this traffic pattern while providing point-to-point routing as an extension. Moreover, since a network typically needs few gateways (often just one), and the gateways need not take part in the sensing process, they can be dedicated devices with much larger memory and computing power. In addition, they can have a tethered power supply and can be connected together with a more reliable communication technology. Consequently, the gateways can take over the point-to-point routing process.

More specifically, in centralized routing, each sensor node maintains a routing entry for one of the gateways, or for \( k \) gateways for fault tolerance \((k\) is constant). Like in shortest-path routing, a routing entry describes the shortest possible route to the gateway. In other words, globally, the nodes are organized in a spanning tree rooted at a gateway (or \( k \) trees), where a routing entry at a node determines the parent of the node in the tree. As a result, when routed to the gateway, packets travel up the tree, along the shortest possible paths. This makes many-to-one routing optimal: its routing stretch is 1.

To also enable routing between any pair of nodes, the gateway supports source routing. All nodes periodically report their neighbors or just their parent in the tree to the gateway. Based on these reports, the gateway computes the shortest path from itself to every other node. When a packet has to be routed from one sensor node to another, it is first forwarded up the tree along the shortest route to the gateway, as determined by the routing entries of the nodes on this route. When it reaches the gateway, its header is augmented with a whole route to the destination node, as computed by the gateway. The addresses of the nodes comprising this route and embedded in the header are then used to forward the packet down the tree, toward the destination node. This algorithm can easily be extended to \( k \) gateways provided that the gateways share their computed paths.

Overall, in centralized routing, despite a linear routing state at a gateway, the amount of state amortized over all nodes remains constant. Moreover, maintaining a linear state at a gateway is possible because gateways are assumed not to be as constrained in terms of memory as sensor nodes. This, combined with the optimal-stretch many-to-one data collection, makes centralized routing
However, it is trivial to observe that the worst-case stretch of centralized routing is very large: it depends linearly on the node population size. In other words, to route to a nearby destination, a source node may have to route through the gateway which may be far away, and this potentially long path has to be traveled twice, up and down the tree, wasting resources. Moreover, in networks with large diameters, the routes from the gateway to the nodes may involve many intermediate hops. Embedding such routes in packet headers may thus involve a lot of overhead. Finally, as the gateway has to route virtually every point-to-point packet, there may be an unnecessary routing hot-spot around the gateway. While a gateway has arguably enough resources to handle its traffic, sensor nodes nearby the gateway may quickly exhaust their energy budgets forwarding messages from and to their subtrees [25]. For these reasons, some sensornet applications may require different routing techniques.

**Geographic Routing**

One such a technique providing constant state without a centralized component is geographic routing [32]. In geographic routing, the routing address of a node corresponds to the geographic coordinates of the node, such as the longitude and latitude. The routing table, in turn, involves one routing entry for each neighbor of the node; the entry for a neighbor contains the geographic coordinates of the neighbor. Therefore, since in large networks the number of neighbors per node is usually considered to be independent of the total node population size, the routing state of a node in geographic routing is considered constant.

When routing a packet to a destination node with certain coordinates, a source and each intermediate node choose as the next hop the neighbor closest to the destination in the geographic coordinate space. This process is often referred to as *greedy forwarding*, as each forwarding node greedily tries to minimize the distance remaining to the destination.

While greedy forwarding is simple and powerful, it has an inherent limitation. There are topologies in which some intermediate node may have no neighbor closer to the destination than the node itself. In other words, *routing voids* exist in such topologies. Upon encountering a routing void, a packet has to be temporarily forwarded to a neighbor farther from the destination than the forwarding node. Choosing such a neighbor, however, requires special mechanisms to ensure that the packet finally reaches the destination, that is, it will not enter an infinite routing loop.

There are a few approaches to effectively handling routing voids. The most
common one, adopted in protocols such as GFG [10], GPSR [61], and the GOAFR+ family [72], involves making the internode connectivity graph planar, that is, removing those entries from the node routing tables which represent crossing edges in the connectivity graph. In such protocols, normally, a packet is routed greedily. However, when it encounters a void, its routing mode is changed from greedy to perimeter. In the perimeter mode, the long-know right-hand graph traversal rule is used to route the packet around the void to gradually reach the destination or at least a node at which greedy forwarding can again make progress. The fact that the connectivity graph is planar ensures the lack of infinite routing loops in the perimeter mode.

However, graph planarization algorithms usually assume that the connectivity graph follows a unit-disk model, which introduces problems when deploying these algorithms in the real world [64]. In a unit-disk model, each node has a fixed circular radio range and can communicate with all and only those nodes which fall within this range. While this model is useful for analyzing routing algorithms, it does not reflect low-power wireless connectivity well. As discussed in Section 2.2 in the real world, phenomena such as communication obstacles, multipath effects, and noise often make the internode connectivity violate the unit-disk model. In effect, a graph planarization algorithm may yield a nonplanar graph, potentially leading to infinite routing loops [64]. The same effect may also be a result of errors in node coordinates; real-world localization techniques are inherently imprecise, and thus, always introduce some errors.

For these reasons, making a connectivity graph planar is challenging in the real world. Kim et al. [64, 63] present a cross-link detection protocol (CLDP) which actively probes internode links and traverses potential voids to remove any crossing links. However, the protocol is relatively complex in that it involves many corner cases, and it is unclear how well it scales in terms of the maintenance traffic necessary for the probes.

To cope with these problems, Leong et al. [84] propose to abandon graph planarization and, instead, to use a spanning tree (or multiple trees) as the fall-back mechanism when greedy forwarding cannot make progress. More specifically, in their protocol, GDSTR, apart from the geographic coordinates of the neighbors, the routing state of a node contains additional routing entries representing the tree: first, like in centralized routing, a routing entry for the parent node in the tree; second, a routing entry for each child node in the tree. Each routing entry for a child node in the tree contains the convex hull over the coordinates of all the nodes in the subtree rooted at this child node. When greedy forwarding cannot make progress at a node, the node checks whether the convex hull of any of its children contains the point with the destination coordinates. If
CHAPTER 2. BACKGROUND

there are such children, the packet is forwarded to the first of them. If there are no such children, the packet is forwarded to the parent. The children repeat this process recursively, so if none of them (or their children, grandchildren, and so on) can make progress, the packet is forwarded up the tree.

While this approach eliminates graph planarization, it introduces other problems. It is not clear, how convex hulls should be represented to both be compact and describe a subtree with a sufficient granularity. Moreover, with a spanning tree, packets destined for nonexisting coordinates always travel through the root node. Consequently, the root node and the surrounding nodes may become routing hot spots. Finally, as GDSTR has not been implemented and tested on real hardware, it still remains to be determined what other practical challenges it has to face.

In any case, irrespective of whether using graph planarization or hull trees, geographic routing results in a large routing stretch. When using graph planarization with the right-hand rule, a packet entering the perimeter mode upon a void may travel around the void in the wrong direction, taking far more routing hops than necessary. While there have been attempts to improve the stretch in such situations [72, 73, 74], it has been proved that the worst-case stretch of geographic routing with local-only information depends linearly on the node population size [72]. Similarly, the worst-case stretch of geographic routing with hull trees is linear with respect to the number of nodes; the proof is the same as for centralized routing.

To reduce the large stretch of geographic routing, protocols which use nonlocal information have been proposed. To route around voids, such protocols either exploit information on the connectivity graph faces which surround voids [30, 86] or use an additional overlay network which imposes additional structure on the connectivity graph [12, 31, 59, 124]. In other words, they trade off routing state for routing stretch.

However, despite promising analytical and simulation results, these protocols have not been implemented and evaluated in the real world. Consequently, it is not clear how practical they are. In particular, many of them involve a complex geometric preprocessing phase of the connectivity graph. It is thus unknown how the protocols adapt to changes in the node population and connectivity.

In general, geographic routing poses significant problems for real-world sensornet deployments. Most of the aforementioned solutions for dealing with routing voids are designed for planar networks and simply do not work in volumetric deployments. Others could, in theory, be ported to volumetric networks, but their implementation complexity would change drastically, and hence, they may be unsuitable for resource-constrained sensor nodes. Volumetric
networks, however, are common in many of the sensornet applications described in Section 2.1. Obtaining node coordinates is also problematic. While solutions such as the Global Positioning System (GPS) may be applicable in outdoor environments with a line-of-sight GPS connection, accurately obtaining node coordinates indoors or in environments with overhead canopy or structures, like tall buildings, is essentially an open problem [80].

**Graph Embedding**

Many of the aforementioned problems of geographic routing stem from the fact that its routing state is based on an artificially imposed coordinate space, and some geometric properties of this space are expected to hold in the internode connectivity graph. In practice, however, due the peculiarities of low-power wireless communication, the internode connectivity in sensornets often violates these properties, which complicates geographic routing.

This observation leads to another routing technique, which abandons artificial geographic coordinates and, instead, uses virtual coordinates synthesized based on the actual internode connectivity. This technique is often referred to as *graph embedding* as it embeds virtual coordinates into a connectivity graph. Such an embedding process can, for example, eliminate routing voids. As a result, simple greedy forwarding may be sufficient for routing.

Rao et al. present a family of protocols that employ iterative relaxation to synthesize node coordinates [108]. In the most general version of their protocols, two nodes are designated as beacons and advertise themselves to all other nodes by flooding the entire network. Based on these advertisements, each node determines if it lays inside or on the perimeter of the network. Each perimeter node then floods the network to advertise itself to all other nodes, which enables each node to compute the distance (the number of hops) to every other perimeter node. Perimeter nodes again flood the entire network, this time broadcasting the computed inter-perimeter distances. Based on the distances, each node computes normalized coordinates for both itself and the perimeter nodes, and the perimeter nodes project their coordinates onto a circle. Finally, all nodes gradually compute their final coordinates with a distributed iterative relaxation algorithm, which works by averaging the coordinates of neighboring nodes.

To route packets, nodes use greedy forwarding in the virtual coordinate space. However, this virtual coordinate system does not ensure a lack of routing voids. Therefore, when a routing void is encountered at a node, the node uses scoped flooding with an expanding radius. More specifically, the node broadcasts the packet to all neighbors, which rebroadcast the packet to their neighbors, and so
on, up to a certain radius. If the destination node is not reached by such a flood, the flood radius is increased, and the flood is restarted.

Since scoped floods are costly, a number of graph embedding protocols aim at synthesizing virtual coordinates that eliminate floods. Leong et al. propose using spring relaxation to synthesize such coordinates [85]. Sarkar et al. demonstrate that conformal mapping with Ricci flows can achieve the same objective [111]. Newsome and Song, in turn, introduce a protocol that supports greedy forwarding in a polar coordinate space built with a spanning tree [100].

While all these protocols demonstrate that geographic coordinates are not necessary to perform geographic routing, they pose many problems for sensor network deployments. Typically, the phase in which they synthesize virtual node coordinates is very long and requires exchanging lots of messages. Moreover, while the protocols need a constant routing state, the state they use to synthesize the coordinates may be even linear with respect to the total node population. Finally, it is not clear how well the protocols react to changes in the node population and connectivity, especially as this may require recomputing the coordinates. Since the protocols have not been implemented and evaluated on real hardware, it is hard to assess the practical significance of such problems.

To the best of my knowledge, the only graph embedding protocol that has been implemented is Beacon Vector Routing (BVR) [35]. In BVR, $B$ nodes are selected as beacons and advertise themselves to all other nodes by flooding the network ($B$ does not depend on the total node population size). Based on the advertisements, each node computes its distance (i.e., the number of hops) to every beacon. Such a vector of distances constitutes the coordinates of the node in the $B$-dimensional beacon space. When routing a packet to a destination, nodes use greedy forwarding in this space. If greedy forwarding fails, nodes try forwarding the packet to the beacon closest to the destination, and they resume greedy forwarding as soon as it can make progress. If the packet reaches the beacon and greedy forwarding still cannot make progress, a scoped flood is performed; the radius of the flood is equal to the number of hops from the beacon to the destination, as determined from the destination address.

While the state maintained by each node in BVR is asymptotically constant, when measured in bytes, it can be quite large [94]. For every neighbor, a node has to maintain $B$ distances from the neighbor to the beacons and $B$ identifiers of these beacons. Moreover, the routing stretch in BVR can be large, especially when scoped flooding is necessary to deliver a packet. Finally, even when scoped flooding is used, it is difficult for BVR to guarantee packet delivery. By and large, the applicability of BVR, like of the other aforementioned constant-state routing protocols, is limited to only some deployment scenarios, in particular, because
of their large worst-case stretch. This warrants investigating alternative routing techniques for sensornets.

### 2.3.3. Compact Routing

One such a technique is compact routing [70]. In contrast to shortest-path routing and constant-state routing, which constitute the two ends of the state-stretch trade-off spectrum of routing techniques, providing either minimal stretch or minimal state, compact routing tries to balance state and stretch. More specifically, Cowen presents a compact routing algorithm that provides a maximum worst-case stretch of 3 with a maximal routing state of $O(N^{\frac{2}{3}})$ entries (where $N$ denotes the total node population size) [19]. Thorup and Zwick improve this scheme by reducing the maximal routing state to $O(\sqrt{N})$ entries while maintaining the maximal stretch of 3 [128]. In other words, in any communication graph, compact routing requires sublinear routing state while ensuring that the worst-case routing stretch is small as well.

Gavoille and Genegler show that no routing algorithm with a maximal stretch smaller than 3 can guarantee a sublinear routing state [40]. Thorup and Zwick, in turn, prove that no routing algorithm with a stretch smaller than 5 can guarantee the worst-case routing state asymptotically smaller than $O(\sqrt{N})$ [127]. These two results mean that the trade-off between routing state and stretch made by compact routing is an optimal one.

The basic idea behind compact routing is designating a number of nodes as beacons, much like in BVR. In contrast to BVR, however, the number of designated nodes is not constant, but proportional to $\sqrt{N}$. Each of the remaining nodes binds itself to the closest beacon, that is, the beacon which it can reach within the fewest hops. The routing address of a node is a concatenation of the unique identifier of the node and the unique identifier of the closest beacon node\(^1\). The routing state, in turn, consists of two types of routing entries: first, one entry for each beacon node, and second, one entry for each non-beacon node that is closer to the node than to its own beacon. The number of entries of both the first and the second type is bounded by the square root of the node population size. Like in shortest-path routing, a routing entry for a node describes the shortest route to that node.

\(^1\)The variant of compact routing described here has been adopted by sensornet implementations, notably by the S4 protocol [94]. It does not strictly match the original compact routing algorithm. Strictly speaking, S4 is not a compact routing protocol, because it does not guarantee the appropriate bound on routing state. However, since the sensornet literature considers the described variant as compact routing, for the sake of simplicity, let us do the same in the remainder of this dissertation.
When routing a packet, a forwarding node first looks up an entry for the destination node in its routing table. If such an entry is found, the packet is forwarded to the destination along the shortest possible path, thereby yielding the optimal stretch of 1. More often, however, the entry for the destination node does not exist. In such a case, the packet is first forwarded along the shortest path toward the beacon closest to the destination node, as determined from the routing address of the destination. As soon as the packet arrives sufficiently close to the destination, that is, at a node that has a routing entry for the destination, it is redirected toward the destination itself. In such a case, the overall routing stretch does not exceed 3.

With their S4 protocol, Mao et al. demonstrate how to make the above algorithm work in sensornets [94]. To this end, they subtly relax the definition of the routing table, such that the theoretical bound on the routing state may no longer hold in arbitrary network topologies. Nevertheless, in their experiments, they show that in typical topologies of sensornets the routing state is indeed proportional to $\sqrt{N}$. Moreover, they present techniques that enable reacting to changes in the node population and connectivity and aiding deployments, such as mechanisms for maintaining node routing tables and addresses, and probabilistic heuristics for electing beacon nodes.

The maximal worst-case stretch of 3, a sublinear state, and the fact that it has been proved to work robustly in sensornet deployments, make compact routing an attractive routing technique for sensornets of moderate sizes. For large networks, however, because of the memory and bandwidth constraints of sensor nodes, the state that grows proportionally to $\sqrt{N}$ may be a major problem. Consequently, for some sensornet applications that require large networks, a routing technique that more significantly reduces its state at the expense of stretch can be of immediate interest.

2.3.4. Hierarchical Routing

A technique that has the potential to significantly reduce state at a small expense of stretch is hierarchical routing. The basic idea behind hierarchical routing is simple: partition nodes into a multi-level hierarchy of areas that enables point-to-point routing with minimal state. The algorithms for hierarchical routing, however, are more complex than for the other techniques surveyed above. Therefore, their detailed description is deferred to the next chapter. Here instead, an overview of the related work is given.

In a pioneering paper on hierarchical routing, Kleinrock and Kamoun show that this routing technique can dramatically reduce node state [66]. More
specifically, they demonstrate that in an optimal area hierarchy the routing state of the nodes can be bounded by $O(\log N)$ entries. Such a small state could ensure virtually unlimited scalability of sensornet deployments.

For this reason, initially, hierarchical routing gained a lot of attention from the sensornet community. In effect, a few hierarchical routing protocols for sensornets have been proposed [16, 22, 75], in addition to several potential applications of the area hierarchy itself [5, 16, 38, 56, 75, 112, 122]. To the best of my knowledge, however, over a decade since their introduction, none of those proposals have been implemented and evaluated on real sensornet hardware.

It can be speculated that one potential reason for this situation is the seeming intricacy of the algorithm; especially, constructing and maintaining an area hierarchy is a complex task [66]. This argument is reinforced by Hagouel, who proves that building an area hierarchy that minimizes the node routing state is an NP-complete problem [42]. Even though an alternative hierarchy, a landmark hierarchy, has been proposed to slightly simplify the maintenance task [131], constructing an optimal landmark hierarchy is closely related to the minimal dominating set [39, p. 190] and minimal $d$-dominating set [3] problems, which have been proved to be NP-complete as well. In other words, only heuristic hierarchy maintenance algorithms are practical. However, due to the resource constraints and distributed nature of sensornets, heuristics for hierarchical partitioning of large graphs developed in other fields, such as in data mining [43, p. 335], are of limited applicability. Similarly, as existing hierarchical routing protocols for sensornets were introduced when sensornet research was still in its initial phase, their hierarchy maintenance mechanisms make several simplifying assumptions that have been later demonstrated to be invalid, such as reliable local broadcast communication or inexpensive scoped floods. Therefore, there is every likelihood that these mechanisms may fail in the real world, which discourages potential implementors.

Another possible reason for the lack of implementation of hierarchical routing may be the fact that the worst-case stretch of this technique can be quite large. For example, in a communication graph with full connectivity (i.e., a clique), the routing stretch of hierarchical routing grows at least logarithmically with the node population size, even for an optimal hierarchy [66]. Such a potentially large routing stretch may negatively affect performance. Therefore, it may be yet another factor that discourages investing effort into implementing and testing such an intricate algorithm as hierarchical routing.

On the other hand, hierarchical routing has the potential to provide excellent performance in sensornets. First and foremost, its polylogarithmic routing state ensures virtually infinite scalability. As such it is extremely attractive for large
networks composed of highly resource-constrained sensor nodes. Moreover, although in arbitrary network topologies hierarchical routing cannot guarantee small worst-case stretch, due to embedding in the physical space, the topologies of sensornets are typically “geometric”: the average number of hops between nodes grows as $N^\nu$ with the node population size (where $0 < \nu \leq 1$). Kleinrock and Kamoun have proved that in such topologies the expected stretch of hierarchical routing can be close to 1 [66]. Finally, appropriate mechanisms for hierarchy construction and maintenance can likely be devised because the resulting hierarchies do not necessarily need to be optimal to support small-state small-stretch routing.

In any case, as illustrated in Figure 2.3, from the routing techniques spectrum surveyed in this chapter, only hierarchical routing does not have an implementation and to date has not been evaluated on real sensornet hardware. With such a gap in the information on the practical performance of the routing techniques spectrum, a sensornet system developer may be forced to make suboptimal choices regarding the routing technology. Therefore, the main objective of this dissertation is to fill in this gap, first, by demonstrating that hierarchical routing is feasible in sensornets, and second, by enumerating its merits and drawbacks.

Figure 2.3: An example illustrating the routing techniques spectrum. Except for hierarchical routing, all routing techniques have already been implemented and evaluated on real sensornet hardware: (a) Hui and Culler [46]; (b) Kim et al. [63]; (c) Fonseca et al. [35]; (d) Mao et al. [94]; (e) Frey and Pind [36]. One of the objectives of this dissertation is to fill in this gap.
Chapter 3

Enabling Hierarchical Routing in Sensornets

Surveying the routing techniques spectrum in the previous chapter, it was argued that while hierarchical routing has the potential to offer excellent performance, devising a hierarchical routing protocol for sensornets poses a number of challenges. First and foremost, the hierarchical routing algorithm is itself quite elaborate. Especially synthesizing and maintaining node routing state requires potentially intricate mechanisms. Moreover, the protocol has to effectively cope with the hardware and deployment constraints of sensornets, in particular, with the memory and bandwidth limitations and the peculiarities of the low-power wireless communication.

This chapter presents PL-Gossip, a practical hierarchical routing protocol for sensornets. It guides the reader through theoretical foundations of the protocol and discusses the decisions taken when designing the protocol. Section 3.1

1The “PL” in PL-Gossip now stands for “Polish” as I am Polish. Originally, it had a different meaning, though.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission of the author and/or a fee.

starts by decomposing a routing protocol for sensornets into well-recognized abstractions reflecting different aspects of routing. Section 3.2 formalizes basic terms and definitions associated with PL-Gossip and hierarchical routing in general. Then, Section 3.3, 3.4, and 3.5 discuss PL-Gossip in terms of each of the abstractions identified earlier. Finally, Section 3.6 offers a concluding discussion regarding the design of PL-Gossip. The proofs of the lemmas proposed in this section can be found in Appendix A.

3.1. ROUTING PROTOCOL SKELETON

Prior research activities and the proposed IP-based network architecture have divided the functionality of a sensornet routing protocol into well-defined abstractions (see Figure 3.1). Each of the abstractions corresponds to a different aspect of routing. Moreover, while some of the abstractions are tightly-coupled to a particular routing technique, here hierarchical routing, others are technique-independent and can often be implemented in various ways.

Starting from the top, packet forwarding is the abstraction responsible for the actual routing process. It covers the functionality for initiating routing a packet at a source node, receiving a routed packet at a destination node, and handling a packet at an intermediate node, from the moment the packet is received until the moment it is passed to a next-hop node (or nodes).

To enable packet forwarding, nodes have to maintain some routing state, which is the task of the routing state maintenance abstraction. In hierarchical routing, the routing state of a node consists of a routing table and a routing address. Nodes have to continuously update their routing tables to account for changes in their population and connectivity. In addition, in PL-Gossip nodes also autonomously synthesize and maintain their routing addresses, which is crucial for the protocol to be self-managed.

Packet forwarding and routing state maintenance both require information on the neighbors of a node, notably the link-layer (MAC) addresses of these neighbors and the information on how reliable the links to these neighbors are. Since low-power wireless connectivity varies in time, maintaining this information has to be performed continuously, which is the goal of the link quality estimation abstraction. While link quality estimation can be implemented solely at the link layer of the protocol stack, the routing layer can typically provide some important feedback [34, 47, 138]. Therefore, the link quality estimation abstraction is usually shared between the routing layer and the link layer.

Before demonstrating in the remainder of this chapter how PL-Gossip
3.2. BASIC TERMS AND DEFINITIONS

The principal idea behind hierarchical routing is organizing nodes into a virtual overlay, a *multi-level hierarchy*, on top of the actual network topology. There are two basic variants of such a hierarchy: an *area hierarchy* [66] and a *landmark hierarchy* [131]. In an area hierarchy, connected nodes are logically clustered into areas, which are then clustered into super-areas, and so on at subsequent higher levels. In a landmark hierarchy, in turn, some nodes are promoted to landmarks, some of the landmarks are promoted to super-landmarks, and so on at subsequent levels; other nodes bind themselves to a landmark at
every level. There are subtle differences between the two types of hierarchies and the corresponding routing algorithms \cite{66,131}; in particular, a landmark hierarchy is considered easier to maintain. \textit{PL-Gossip}, however, can work with any of these hierarchy types. In support of this claim, this chapter presents a variant of \textit{PL-Gossip} using an area hierarchy, while the following chapters switch to a landmark hierarchy.

### 3.2.1. Area Hierarchy Model

To enable routing, a multi-level area hierarchy has to ensure that between any two nodes in an area there exists a path that consists only of nodes belonging to this area. However, to provide small routing state and to facilitate maintenance, the hierarchy should also have additional properties. For this reason, for the variant of \textit{PL-Gossip} presented in this chapter, a custom area hierarchy is devised.

To formally define the properties of the hierarchy, the internode connectivity is modeled as an undirected graph. A wireless link is assumed to exist from node $A$ to node $B$ if and only if node $B$ is able to receive datagrams (i.e., messages in the terminology of the link-layer) transmitted by $A$. Since wireless low-power communication is often unreliable and asymmetric, for each node $A$ and $B$, the link quality estimation component measures the quality of a link from $A$ to $B$ and from $B$ to $A$, as discussed in Section \ref{sec:link_quality}. Nodes $A$ and $B$ are considered \textit{neighbors} by \textit{PL-Gossip} if and only if there exist high-quality wireless links

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{network_example.png}
\caption{An example of a connected network. The straight solid lines represent the neighbor relation. Two nodes are considered neighbors if and only if there exists a high-quality bi-directional link between them.}
\end{figure}
between them in both directions. The graph reflected by the neighbor relation is thus undirected. It is also assumed to be connected, that is, the network is assumed not to be partitioned. Moreover, in this dissertation, it is assumed that nodes are immobile. Although in principle the mechanisms PL-Gossip introduces to handle changes in the node population and connectivity can support limited mobility, a short radio range of sensor nodes makes handling highly mobile networks extremely challenging; such networks constitute a separate research area [24]. Practical solutions to the mobility problem typically involve some static backbone through which mobile nodes communicate with each other. PL-Gossip can be a foundation of such a backbone. Figure 3.2 presents a sample network with the neighbor relation between nodes depicted as solid straight lines.

Based on their connectivity, reflected in the neighbor relation, nodes are grouped into sets, called clusters. The clusters correspond to network areas and form a recursive multi-level hierarchy (see Figure 3.3). Two clusters are considered as adjacent if and only if they contain two nodes (one in each of them) that are neighbors. For example, in Figure 3.3 level-0 cluster \{P\} (the white ellipse surrounding node P) is adjacent to the following level-0 clusters: \{O\}, \{R\}, and \{J\}. Likewise, level-1 cluster \{O,P,R\} (the light gray area containing nodes O, P, and R) is adjacent to the following level-1 clusters: \{M,N\} and \{G,H,I,J,K,L\}; but not to level-1 cluster \{A,B,C,D,E,F\}.

The variant of PL-Gossip presented in this chapter defines the following four
custom properties for a cluster hierarchy:

**Property 1** Level-0 clusters correspond to individual nodes.

**Property 2** There exists a single, level-$H$ cluster that contains all nodes. This cluster is referred to as the top-level cluster.

**Property 3** Level-$i+1$ clusters (where $0 \leq i < H$) are composed out of level-$i$ clusters, such that each level-$i$ cluster is contained in exactly one level-$i+1$ cluster. A level-$i+1$ cluster is the supercluster for its level-$i$ clusters, and conversely, these level-$i$ clusters are the subclusters of their level-$i+1$ cluster.

**Property 4** Each level $i+1$ cluster (where $0 \leq i < H$) contains at least one subcluster that is adjacent to all other subclusters of this cluster. One of such subclusters is referred to as the central subcluster.

To prove that a hierarchy satisfying the above properties can enable hierarchical routing, one has to prove the following lemma, which essentially formalizes the aforementioned necessary condition for hierarchical routing in an area hierarchy. This proof can be found in Appendix A.1.

**Lemma 3.1** For each cluster and any two nodes, $A$ and $B$, belonging to this cluster, there exists a path between $A$ and $B$ in the graph reflected by the neighbor relation that consists only of nodes belonging to the cluster.

In fact, not only can one prove that a path between any two nodes in a cluster exists within the cluster, but one can also derive an upper bound on the length of such a path. To illustrate this, let us first recursively define the *head* node of a level-$i$ cluster:

1. if $i = 0$, then the head of the cluster is the sole node constituting the cluster;
2. if $i > 0$, then the head of the cluster is equal to the head of the central subcluster.

A node is a *level-$i$ cluster head* if and only if it is the head of some clusters at levels from 0 to $i$, but not the head of any level-$i+1$ cluster. In Figure 3.3, for instance, node $P$ is the head of level-0 cluster $\{P\}$, and likewise all other nodes are the heads of their own level-0 singleton clusters. Similarly, consider level-1 cluster $\{A,B,C,D,E,F\}$. The only candidate for the central subcluster of this cluster is level-0 cluster $\{A\}$, as $\{A\}$ is adjacent to all the remaining subclusters $\{B\}, \{C\}, \{D\}, \{E\}$, and $\{F\}$. Consequently, node $A$ is the head of level-1 cluster.
\{A, B, C, D, E, F\}. In contrast, in cluster \{O, P, R\}, any of the subclusters (\{O\}, \{P\}, or \{R\}) can be the central subcluster, thus any of nodes O, P, and R can be the head of this level-1 cluster. Finally, in the top-level cluster covering all nodes, any of subclusters \{G, H, I, J, K, L\} and \{M, N\} can be the central subcluster. Therefore, reasoning recursively, any of nodes L, M, and N can be the head of the top-level cluster.

The following bounds on path lengths between nodes hold in the custom hierarchy satisfying Properties 1–4. The proofs of these bounds and the fact that the bounds are tight (i.e., no better bounds are possible) can be found in Appendix A.2, A.3, and A.4 respectively.

**Lemma 3.2** A node from a level-i cluster can reach some node in any adjacent level-i cluster in at most $3^i$ hops in the graph reflected by the neighbor relation.

**Lemma 3.3** The distance between the head nodes of two adjacent level-i clusters is at most $3^i$ hops in the graph reflected by the neighbor relation.

**Lemma 3.4** The distance between any two members of a level-i cluster is at most $3^i - 1$ hops in the graph reflected by the neighbor relation.

The above bounds effectively mean that cluster diameters can grow exponentially with the hierarchy level. In other words, the number of levels ($\mathcal{H} + 1$) in a hierarchy satisfying Properties 1–4 can depend polylogarithmically on the number of nodes in the network. This, in turn, gives the hierarchy the potential to offer polylogarithmic routing state, as is discussed later and is demonstrated experimentally in the next chapter.

Note, however, that there is neither a property that would guarantee that a cluster has more than one subcluster nor a property that would bound the number of subclusters in a cluster. This means that while the four properties make polylogarithmic routing state possible, they do not guarantee it. Although it is possible to add such properties, maintaining a hierarchy that would satisfy the resulting property set would add complexity to the PL-Gossip protocol. More specifically, ensuring that a cluster has at least two subclusters requires special cluster rebalancing mechanisms. Such mechanisms, however, have to guarantee that cluster membership does not thrash, which is not trivial, especially on resource-constrained sensor nodes. Therefore, this chapter presents a variant of PL-Gossip that maintains just Properties 1–4, especially since the presented variant performs well in practice. As mentioned in a number of places further in this chapter, there are many such design decisions and trade-offs between, for instance, protocol performance and complexity, that one can make. For the sake
of clarity, however, this chapter attempts to always present the simplest illustrative variant of \textit{PL-Gossip}, while enumerating possible improvements.

### 3.2.2. Hierarchical Routing Address

To illustrate how hierarchical node addressing works in the custom hierarchy defined above, in the remainder of this section it is assumed that in Figure 3.3, the central subcluster of level-1 cluster \{\texttt{O,P,R}\} is level-0 cluster \{\texttt{O}\}, the central subcluster of level-1 cluster \{\texttt{M,N}\} is level-0 cluster \{\texttt{M}\}, the central subcluster of level-1 cluster \{\texttt{A,B,C,D,E,F}\} is level-0 cluster \{\texttt{A}\}, the central subcluster of level-1 cluster \{\texttt{G,H,I,J,K,L}\} is level-0 cluster \{\texttt{L}\}, and the central subcluster of level-2 cluster covering all nodes is level-1 cluster \{\texttt{G,H,I,J,K,L}\} (see Figure 3.4). Consequently, node \texttt{L} is a level-2 cluster head, nodes \texttt{O}, \texttt{A}, and \texttt{M} are level-1 cluster heads, and all remaining nodes are level-0 cluster heads.

Assuming that each node is given a unique identifier, a cluster can be uniquely identified by its level and the unique identifier of its head. Notation \(C^i_V\) is used to denote a level-\(i\) cluster with head node \(V\). For example, as depicted in Figure 3.4, \(C^0_P = \{P\}\), \(C^1_O = \{O,P,R\}\), and \(C^2_L\) is the set containing all nodes.

The \textit{label} of a node is defined as a concatenation of the identifiers of the heads of all the clusters the node belongs to, starting from level 0 up to level \(\mathcal{H}\). For instance, the label of node \texttt{P} in the sample hierarchy from Figure 3.4 is \texttt{POPL}\(=\texttt{P}\).
3.2. Basic Terms and Definitions

Figure 3.5: The label tree for the hierarchy from Figure 3.4. Singleton level-0 clusters are omitted for clarity. The dashed lines reflect the parent-child relationship between cluster heads at subsequent levels. The label of a node can be obtained by concatenating node identifiers on the path from the leaf representing the node to the root of the tree.

$L(P) = P.O.L$ because: at level 0, node $P$ belongs to the singleton cluster of which it is the head ($C_0^P$); at level 1, $P$ belongs to the cluster with head node $O$ ($C_O^1$); at level 2, $P$ belongs to the cluster with head node $L$ ($C_L^2$). The label of node $O$, in turn, is $L(O) = O.O.L$, as $O$ belongs to the following clusters: $C_O^0$, $C_O^1$, and $C_O^2$. Finally, the label of node $L$ is $L(L) = L.L.L$, as $L$ belongs to clusters $C_L^0$, $C_L^1$, and $C_L^2$. In general, the label of a level-$i$ cluster head, $V$, has $V$ at all positions from 0 to $i$. The label of a node constitutes the hierarchical routing address of this node.

Node routing addresses can be represented as an $H+1$-level tree, like in Figure 3.5. A node in the tree corresponds to some node in the network topology, more precisely, to a node acting as the head of some cluster. Thus, leaf nodes in the tree correspond to the heads of level-0 singleton clusters, their parents in the tree — to the heads of the superclusters, that is, level-1 clusters, and so on, such that the root node of the tree corresponds to the head of the top-level cluster. In this representation, the label of a node is obtained by concatenating the identifiers of cluster heads on the path from the leaf representing the node in the tree to the root of the tree. The tree representation of node labels is useful when analyzing the algorithms PL-Gossip introduces for maintaining the cluster hierarchy.

3.2.3. Hierarchical Routing Table

Based on its hierarchical routing address (the label), each node maintains a hierarchical routing table. The routing table of a node contains a number of
Figure 3.6: The routing table of node $P$ from Figure 3.4. Singleton level-0 clusters are omitted for clarity. The dashed lines reflect the parent-child relationship between cluster heads at subsequent levels. At each level $i$, the routing entries of a node correspond to sibling level-$i$ clusters of the level-$i$ cluster the node belongs to.

routing entries for each level of the hierarchy. Each level-$i$ routing entry of a node corresponds to one level-$i$ cluster that is a sibling in the hierarchy of the level-$i$ cluster the node belongs to (see Figure 3.6). For node $P$, for example, the sibling clusters of level-0 cluster $C_0^P$ are clusters $C_0^O$ and $C_0^R$, as these clusters are contained in the same level-1 cluster, $C_1^O$, as cluster $C_0^P$. Consequently, at level 0, node $P$ maintains two routing entries: one for $C_0^O$ and one for $C_0^R$. Likewise, at level 1, $P$ belongs to $C_1^O$, for which the sibling clusters are $C_1^A$, $C_1^L$, and $C_1^M$. Therefore, at level 1, node $P$ (and, similarly, any node in $C_1^O$) has one routing entry for each of these clusters. If the hierarchy had more levels, the routing entries at these levels would be maintained according to the same rules. For hierarchy maintenance purposes, apart from the entries for sibling clusters, a node in $PL$-Gossip also stores one entry for each of its own clusters, thus $H+1$ additional entries in total.

In general, the number of entries in a hierarchical routing table of a node is sublinear with respect to the node population size. For example, while the size of the node population in Figure 3.5 is 17, the routing table of node $P$ has only 8 entries, the routing table of node $I$ has 11 entries, and the routing table of node $M$ has 7 entries. As discussed in the previous chapter, Kleinrock and Kamoun have shown that the size of the routing table of a node can depend polylogarithmically on the node population size [66]. In general, however, the size of the routing table depends to the algorithms employed for constructing and maintaining the cluster hierarchy. For this reason, the next chapter studies the routing table sizes offered
by PL-Gossip.

A routing entry for a level-$i$ cluster maintained by PL-Gossip at a node consists of several fields, as listed in Listing 3.1 and in the examples in Table 3.1. Firstly, the entry contains the level, $i$, of the cluster and the unique identifier of the head of the cluster, which together allow for uniquely identifying the cluster and for looking the entry up in the routing table. Secondly, to enable routing, the entry also contains a list of next-hop neighbors on the shortest path from the node to that head as well as the number of hops on this path. Moreover, to allow for ensuring Property 4, the entry contains a bit indicating whether the cluster it refers to is adjacent to the level-$i$ cluster of the node. Finally, the entry contains some additional fields (not listed in Table 3.1), notably the last sequence number generated by the head of the cluster it refers to and the time-to-live for this sequence number. Such fields are used for maintaining the node routing tables in the presence of network dynamics, for instance, for removing entries for no longer existing or unreachable clusters.

```plaintext
Listing 3.1: The most important fields of a routing entry. Some maintenance fields and fields related to the selection of the best next-hop neighbor during packet forwarding are omitted for brevity.

```
Table 3.1: Entries in the routing tables of sample nodes from Figure 3.3. Each routing entry at a node corresponds to some cluster and is represented by a row in the routing table of the node. The columns of the table, in turn, denote particular fields of the entry: column “L” denotes the level of the cluster the entry refers to; column “H” denotes the unique identifier of the head of this cluster; column “N” denotes the link-layer addresses of possible next-hop neighbors on the shortest path to the head (for clarity, just one neighbor is shown); column “D” denotes the number of hops on this path; column “A” denotes whether the cluster the entry refers to is adjacent to the same-level cluster of the present node. The entries for the own clusters of the nodes are again omitted for brevity.

Given the above hierarchical addresses and routing tables, hierarchical routing becomes straightforward. Essentially, it corresponds to forwarding a packet to clusters in which node addresses match longer suffixes of the destination address; the matching starts from the minimal-level at which the destination address differs from the source address and continues down to level 0, matching which denotes
that the packet has reached its destination. This process will be illustrated using code listings and by means of an example.

### 3.3.1. Forwarding Algorithm

Each routed packet has a header that contains the routing address of the destination and, optionally, of the source node. These are filled in by the source node when it initiates routing. In addition, the header often contains a time-to-live (TTL) for the packet, which denotes the number of hops the packet is still allowed to travel, and the goal of which is to help detecting and breaking routing loops in the presence of node failures. The source node can compute the TTL for a packet based on the address of the destination node and its own address. More specifically, if $i$ denotes the minimal level at which the source and destination have the same cluster head identifier in their routing addresses, the packet should be able to reach the destination within $3^i - 1$ hops, as guaranteed by Lemma 3.4.

For example, the routing address of node $P$ from Figure 3.3 is $P.O.L$ while the address of node $C$ is $C.A.L$ (cf. Figure 3.5). The minimal common level of these two nodes is thus 2, which corresponds to cluster $C^2_0$ to which both $P$ and $C$ belong (cf. Figure 3.5). Therefore, from Lemma 3.4, the TTL of a packet from $P.O.L$ to $C.A.L$ is $3^2 - 1 = 8$ hops. In practice, the TTL may be set to a slightly higher value, to account for potential disturbance in the network topology due to changes in the node population or connectivity. Listing 3.2 presents the $PL-Gossip$ header initialization routine for a routed packet. Note that the packet header may involve additional fields, but they are omitted here for brevity.

When the packet has been initialized, the following forwarding activities are performed. The packet is placed in a queue until the moment it can be forwarded. When it is time to forward it, the routing protocol analyzes the packet header and the routing table of the node to obtain a list of potential next-hop neighbors to which the packet can be forwarded. One such a neighbor is chosen, and the packet is transmitted to this neighbor. However, since low-power wireless communication is unreliable, the packet may get lost. To recover from such failures, hop-by-hop acknowledgments can be employed. If the neighbor acknowledges the reception of the packet, the forwarding process for the packet finishes at the present node. If the reception has not been acknowledged, the packet may have been lost and thus some recovery should be performed. To this end, the node puts the packet back into the forwarding queue. It then backs off for some period before attempting retransmission of the packet, which essentially allows for coping with temporal packet loss due to collisions or noise. After the back-off period, the packet is retransmitted, potentially to a different
function initRoutingPktHdr(hdr : RoutingPktHdr, dstAddr : RoutingAddr);

var
i : integer;

begin
hdr.srcAddr ← this.addr;
hdr.dstAddr ← dstAddr;
i ← findCommonCluster(this.addr, dstAddr);
hdr.ttl ← intpow(3, i) - 1;
end; { function initRoutingPktHdr }

function findCommonCluster(addr1 : RoutingAddr, addr2 : RoutingAddr) : integer;

var
i : integer;

begin
i ← 0;
while i < min(addr1.len, addr2.len) do begin
if addr1[i] = addr2[i] then break;
i ← i + 1;
end;
return i;
end; { function findCommonCluster }

Listing 3.2: The routing packet header initialization function. It stores the routing address of the source and the destination and computes the TTL based on Lemma 3.4.

The recovery process is repeated until some neighbor acknowledges the reception of the packet, or the number of retransmissions exceeds some threshold. Essentially, the same process is repeated at the next-hop neighbor upon reception of the packet, and, in general, at each intermediate node forwarding the packet.

The above common packet forwarding mechanisms have been selected because, apart from being suitable for illustrative purposes, they perform well in practical sensornet deployments, offering as much as 98% end-to-end best-effort packet delivery rates in multihop networks [46, 47]. However, packet forwarding has received a lot of research attention. Essentially, each of the activities it involves poses some challenge, either generic to routing or specific to sensornets: stemming from the resource constraints of sensor nodes or the unreliability of low-power wireless communication. Examples of such research problems include: admitting packets to the forwarding queue [33, 116], recovering from missing routing entries due to changes in the network topology [47, 94], selecting the next-hop neighbor from a list of candidates [118, 138], efficiently detecting packet loss [47, 118, 138], choosing a back-off period for retransmissions [119], or abandoning recovery after packet loss, and instead, using opportunistic techniques [41, 113, 139]. Despite advancements in these areas, the functionality
expected by the packet forwarding component from any routing technique remains largely unchanged: deliver a list of potential next-hop neighbors based on the header of a packet and the routing table of the node. Consequently, this functionality will be the focus of the remainder of this section.

To obtain the list of potential next-hop neighbors, the node forwarding a packet compares the destination address embedded in the packet with its own routing address (see Listing 3.3). Let \( i \) denote the minimal level at which both the addresses have the same cluster head identifier, be it \( V \), which means that both, the forwarding node and the destination node, belong to the same level-\( i \) cluster, \( C_V^i \). This also means that they belong to the same level-\( j \) clusters for all \( j \geq i \); in other words, their addresses match starting from level \( i \). The principal idea behind hierarchical routing is forwarding a packet toward clusters in which the addresses of the subsequent forwarding nodes will match longer suffixes of the destination address, such that ultimately the address of some forwarding node will match the whole address of the destination node, which means that the destination has been reached. In line with this idea, the current forwarding node looks up in its routing table an entry for that subcluster of \( C_V^i \) that contains the destination node. This is a level-\( i-1 \) cluster with the head identifier equal to the \( i-1 \)-st element of the destination address (cf. line 14 of Listing 3.3). The organization of the routing table and Lemma 3.1 guarantee that such an entry always exists if there are no failures in the system (cf. Figure 3.6). The list of potential next-hop neighbors is equal to the next-hop neighbors associated with the found routing entry. One of these neighbors is chosen and the packet is forwarded to this neighbor, as described above. The address of the neighbor can again match from level \( i \) or from some lower level, which means that the level-\( i-1 \) cluster containing the destination has been reached. In any case, the neighbor repeats the process and so on, until the level-0 singleton cluster containing the destination node (the destination node itself) has been reached.

Let us illustrate a whole sample routing process by means of an example from Figure 3.7. In the example the routing tables shown in Table 3.1 are used, hence, it is assumed that there is at most one next-hop neighbor associated with any routing entry. This eliminates the need for additional neighbor selection from the list of potential candidates.

A source node, \( P \), with address \( P.O.L \) needs to route a packet to a destination node, \( C \), with address \( C.A.L \). To forward the packet, \( P \) compares its routing address with the destination address. The addresses match starting from level 2. Therefore, \( P \) looks up in its routing table an entry for the level-(2−1) cluster containing the destination. From the destination address, \( C.A.L \), this cluster is equal to \( C_A^1 \). As shown in Table 3.1, the next-hop neighbor associated with the
function lookupNextHopCandidates(
    hdr : RoutingPktHdr : NextHop[];
var
    i : integer;
    rtEntry : RoutingEntry;
begin
    i ← findCommonCluster(this.addr, hdr.dstAddr);
    if i = 0 then
        return [this.meAsNextHop]
    else begin
        rtEntry ← lookupInRoutingTable(i - 1, hdr.dstAddr[i - 1]);
        if rtEntry ≠ null then
            return rtEntry.nextHopNeighbors
        else
            return []
    end;
end;

function lookupInRoutingTable(
    level : integer, headID : integer) : RoutingEntry;
Listing 3.3: The next-hop lookup routing function for an area hierarchy. It finds the minimal-level cluster toward which the packet can be forwarded. This is the subcluster containing the destination node of the cluster that contains both the present node and the destination node. A list of next-hop neighbors on the shortest paths to the head of such a subcluster is returned (as associated with the routing entry for this subcluster). The above function assumes a lack of failures; it gets more intricate when failures may occur.

entry node $P$ maintains for $C^1_A$ is node $R$. The packet is thus forwarded to node $R$.

Likewise, node $R$ compares its address, $R.O.L$, with the destination address, and again, since the addresses match just from level 2, the target cluster is $C^1_A$. Therefore, $R$ forwards the packet to the neighbor associated with the entry $R$ maintains for $C^1_A$, that is, to node $M$ (cf. Table 3.1). Node $M$ is in the same situation as nodes $P$ and $R$, and, as a result, it forwards the packet to node $B$ (cf. Table 3.1).

Reaching node $B$, however, means that the level-1 cluster containing the destination has been reached: the address of node $B$, that is, $B.A.L$, matches the destination address, $C.A.L$, at level 1. Therefore, to continue the routing process, node $B$ looks up in its routing table an entry for the level-$(1-1)$ cluster containing the destination, that is, for $C^0_C$. The next-hop neighbor associated with this entry is node $A$ (cf. Table 3.1), to which the packet is forwarded. Node $A$ repeats the process and, as a result, the packet reaches node $C$. At this point, the address
3.3. Hierarchical Packet Forwarding

The routing is performed by forwarding a packet toward clusters in which the addresses of the subsequent forwarding nodes will match longer suffixes of the destination address.

of the present node and the address of the destination node match from level 0, which means that the destination has been reached.

3.3.2. Remarks

In the above example, the packet reached the destination in 5 routing hops. While the length of this routing path is smaller than the 8 hops computed as the TTL for the packet, the routing path is not optimal. As depicted in Figure 3.8, the optimal path has 4 hops and differs from the routing path taken starting from node $M$. To route within 4 hops, node $M$ in the above example should have forwarded the packet to node $N$ instead of node $B$. However, as can be observed in Table 3.1, node $M$ does not have any information associated with the routing address $C.A.L$ that would be related to node $N$ as the next routing hop: $M$ does not maintain any entry for cluster $c^0_L$, which is consistent with the definition of a hierarchical routing table; the entry it maintains for cluster $c^1_A$ refers to node $B$ as the next hop; and the entry it maintains for cluster $c^2_L$ refers to node $I$ as the next hop. This is because the hierarchical network organization aggregates routing information by storing pointers to selected clusters rather than all nodes in the network. Although this reduces the number of routing entries at a node, it increases the routing stretch. In the above example, the stretch of the routing path measured in hops, the *hop stretch*, is equal to $5/4 = 1.25$. The routing stretch offered by PL-Gossip is studied in the next chapter.

![Figure 3.7: Routing a packet from node P.O.L to node C.A.L. The routing is performed by forwarding a packet toward clusters in which the addresses of the subsequent forwarding nodes will match longer suffixes of the destination address.](image-url)
CHAPTER 3. ENABLING HIERARCHICAL ROUTING IN SENSORNETS

Figure 3.8: The stretch of the routing path from Figure 3.7. The routing path taken from node P.O.L to node C.A.L by hierarchical routing consists of 5 hops. In contrast, the optimal path involves just 4 hops. Therefore, the hop stretch of the routing path is 1.25. This hop stretch above 1 is a result of the aggregation of route information due to clustering, and is an inherent feature of routing techniques with sublinear routing state.

Note also that, even though a packet is routed toward the heads of the clusters containing the destination, the heads do not necessarily forward the packet as it can be redirected earlier toward lower-level heads. While in the above example the level-1 cluster head, A, did route the packet for node C, it had to do it because it is present in all paths from node B to node C that consist only of nodes from $C_A^1$. In contrast, the top-level cluster head, L, did not route the packet. Likewise, if a packet is routed from node P.O.L to node E.A.L (see Figure 3.9), node A does not need to route the packet because there exists a path from B to E that does not involve A (cf. the routing table of node B in Table 3.1). Consequently, while initially a packet from P.O.L to E.A.L is routed toward A, it is redirected toward E before reaching A. The fact that cluster heads do not need to route packets destined for nodes in their clusters implies that the heads need not be inherent routing hot spots, that is, they are not necessarily routing more traffic than other nodes.

Finally, to initiate the routing process, a source node has to know the routing address of the destination node. Considering the solutions to this problem adopted in the Internet, I argue that the address resolution functionality is beyond the scope of a routing protocol. In today’s Internet, for example, resolving a human-readable host name to an IP address used for routing is performed by the Domain Name Service (DNS). Similar solutions may thus be
adopted for hierarchical routing. In particular, a few proposals for distributed address resolution use the area hierarchy itself to implement a distributed store-and-look-up system for hierarchical node addresses [16, 22, 75, 94]. For these reasons, the address resolution problem is not studied in this dissertation.

3.4. ROUTING STATE MAINTENANCE

While hierarchical packet forwarding is itself relatively straightforward, the major problem a hierarchical routing protocol has to address is constructing and maintaining a cluster hierarchy, that is, the node routing addresses and routing tables. As speculated in the previous chapter, the complexity of this problem is a likely reason that none of the hierarchical routing protocols proposed to date has been implemented for sensornets. PL-Gossip demonstrates how to make the hierarchy maintenance problem tractable in the real world.

3.4.1. Principal Idea

To make the hierarchy maintenance problem tractable, PL-Gossip follows a practical approach. Rather than trying to provide a nearly-optimal hierarchy, it
aims to provide just a best-effort one — for illustrative purposes of this chapter — a hierarchy that satisfies Properties 1-4. Instead however, the hierarchy maintenance mechanisms in *PL-Gossip* are required to address important practical aspects. First, they have to be **robust**: they have to be able to recover after changes in the node population and connectivity; they also have to work with unreliable communication. Second, they have to be **self-managed**: using these mechanisms, nodes have to be able to maintain both their routing tables and routing addresses, without user intervention; little or no manual configuration should be necessary for the mechanisms to work. Third, the hierarchy maintenance mechanisms have to be **self-contained**: they must not assume that certain properties of protocols or applications running on top will be used for hierarchy maintenance purposes; instead, they have to be able to build and maintain the hierarchy even when they run in isolation from other protocols. Finally, the mechanisms should be **efficient**, more specifically, they should not involve solutions that have been already proved to be inefficient in sensor networks. For example, multi-level scoped flooding, which has been adopted by many early proposals of hierarchical routing protocols [5, 22, 75, 122], is considered as a highly-inefficient mechanism [89, 101]. In general, the mechanisms used for maintaining the cluster hierarchy in *PL-Gossip* emphasize these practical aspects to broaden the range of potential applications of the protocol and to facilitate using the protocol in the real world.

To this end, these hierarchy maintenance mechanisms are based on a combination of two simple concepts: **asynchronous local gossiping** and **local operations**. More specifically, nodes operate in rounds, each lasting $T$ time units; the rounds of different nodes do not need to be synchronized. In every round, in a heartbeat message, each node broadcasts its routing state, that is, its routing address (the hierarchical label) and routing table. The message is received by the neighbors of the node (i.e., the nodes within the radio range of the node). The neighbors subsequently merge the received state with their own local state. Likewise, they broadcast their state once per round. Such repeated asynchronous exchange (gossiping) and merging of local node routing state enables the hierarchy information to implicitly propagate throughout the network over multiple hops. In particular, nodes learn about clusters and cluster heads in their vicinity.

Based on this knowledge, the nodes construct and maintain the cluster hierarchy. Hierarchy construction is performed in a bottom-up fashion using probabilistic heuristics. Nodes probabilistically promote themselves to higher-level cluster heads by locally modifying their hierarchical labels, effectively spawning higher-level clusters. When they gossip heartbeat messages
in subsequent rounds, their neighbors, the neighbors of these neighbors, and so on learn about the newly created clusters, and can join those clusters also by modifying their labels locally. In this way, the nodes gradually construct the cluster hierarchy. Hierarchy repair after detecting a failure of a node or a connectivity change is performed using the same mechanisms. Detecting a node failure, in turn, is also relatively easy as the failed node simply stops gossiping its state; the same applies for a change in the internode connectivity. Therefore, in general, any modifications to the cluster hierarchy are performed locally and are propagated throughout the network by gossiping.

The above mechanisms are inherently robust, self-managed, and self-contained. Starting from the end, their self-containment stems from the fact that they cover all aspects of hierarchy maintenance: construction, failure detection, and failure recovery; in none of these aspects do they depend on other protocols or some peculiar properties of the application. Their self-management is apparent as well: the whole process of hierarchy maintenance is performed autonomously by the nodes; the mechanisms involve only one parameter that has to be configured by the user, that is, the global round length, $T$. Their robustness to failures manifests itself in the automatic recovery from changes in the node population and connectivity. Moreover, there are only a few possible topology changes that affect many nodes, such as a failure of the top-level cluster head, and the great majority of changes affect few nodes, and thus require local-only repair activities. By varying the round length, $T$, one can explore the trade-off between the latency of reacting to changes and resource consumption. Finally, robustness to packet loss is a consequence of the periodic nature of gossiping: since in each round a node gossips its whole routing state, there is some redundancy in the data exchanged in consecutive rounds; thus, even if some neighbor misses some of the heartbeat messages issued by a node, it will likely receive the data in one of the subsequent rounds. In addition, the mechanisms are also not apparently inefficient. Even considering its redundancy, gossiping is more efficient at disseminating information than flooding [47, 89]. All in all, the mechanisms introduced by $PL$-$Gossip$, combining asynchronous local gossiping and local operations, address the practical aspects of cluster hierarchy maintenance.

In the remainder of this section, these mechanisms are explained in detail.

3.4.2. Routing Table Maintenance

As described in Section 3.2.3, the routing table of a node contains at every level of the hierarchy an entry for the cluster the node belongs to at this level and one entry for each sibling cluster at this level. A routing entry for a level-$i$ cluster consists
of the level, \( i \), of the cluster, a bit indicating whether the cluster is adjacent to the same-level cluster of the node, the identifier of the head of the cluster, a sequence number generated by the head, a time-to-live (TTL) for this sequence number, a list of next-hop neighbors on the shortest path to the cluster head, and the number of hops on this path (hop count).

The routing entries are continuously maintained by the nodes. For instance, when a cluster head fails, an entry for the corresponding cluster has to be evicted from all the routing tables that contain it. Likewise, when a new neighbor of a node is added to the network or some of the existing neighbors are no longer reachable due to a communication obstacle, the next-hop list of some entries may need to be updated, either due to a new better path becoming available or due to an existing path having deteriorated. Such maintenance of routing entries is performed with a custom hierarchical distance-vector algorithm.

A routing entry for a cluster originates at the head of the cluster, which generates a new sequence number for the entry, zeroes the hop count, and sets the adjacency bit. A new sequence number for the entry is generated at the end of each gossiping round. Such a refreshed entry, together with all other routing entries maintained by the cluster head, is embedded in the next heartbeat message of the head. More specifically, for each entry from the routing table, the following fields are stored in a corresponding record in the heartbeat message: the level, the identifier of the corresponding head, the last sequence number, the hop count, and the adjacency bit. When the head broadcasts the heartbeat message, its neighbors can refresh (or create) their routing entries corresponding to the cluster of the head (as well as the entries corresponding to other records embedded in the message). When they broadcast their heartbeat messages, their neighbors can also refresh their entries for the cluster, and so on.

Merging the local routing table of a node with the routing table received from a neighbor is performed like in a standard distance-vector algorithm, but with the exception that the node considers only a subset of the received routing records. More specifically, when node \( A \) receives a heartbeat message from node \( B \), it determines the minimal level of a cluster it shares with \( B \) by comparing its label with the label of \( B \) from the message. If the minimal level is \( i \), \( A \) can update its routing table with those entries from the routing table of \( B \) that are in rows not lower than \( i-1 \). In contrast, if there is no common cluster for \( A \) and \( B \) (Property 2 is violated), \( A \) opportunistically updates its routing table by adding entries for those clusters of which \( B \) is member and which are at level \( \text{len}(L(A)) - 1 \) and above, where \( \text{len}(L(A)) \) is the length of the label of \( A \). This latter case allows \( A \) to propagate information about the hierarchy property violation among the members of its cluster, which is necessary when constructing the hierarchy and recovering
3.4. ROUTING STATE MAINTENANCE

from failures.

When updating a routing entry for a cluster, an objective of a node is to choose as the next hop those neighbors whose entries for this cluster, first, have the adjacency bit set, and second, like in a distance-vector algorithm, minimize the path to the head of the cluster. A simplified update algorithm for a routing entry is presented in Listing 3.4. The algorithm becomes more complicated when multiple neighbors are allowed as the next-hop candidates and when the freshness of the received routing records is considered when updating the local entries of a node. For such an algorithm the reader should refer to the source code, the overview of which is presented in Appendix C. In short, the algorithm guarantees freshness of the route information, while correctly propagating the adjacency bit and ensuring shortest paths to the cluster heads. It also controls the time-to-live of an entry, by setting it upon refreshing the entry either to the number of hops to the head or, more conservatively, to a value computed from Lemma 3.4.

If a node has not refreshed a routing entry for a certain number of rounds, that is, the time-to-live for this entry has expired, the node concludes that the cluster represented by the entry is no longer reachable, for instance, because the cluster head has died or all the paths to the head have been broken. Such an unreachable entry should be removed from the routing table of the node. Simply removing the entry, however, might result in routing cycles. Therefore, to prevent routing cycles when node failures and connectivity changes occur, PL-Gossip uses route poisoning: before removing an entry a node explicitly marks it as unreachable. Such an entry is broadcast in the heartbeat messages of the node for several rounds, which allows other nodes to learn about the failure as well. Only an entry with a fresher sequence number is allowed to refresh a poisoned entry; such a situation means that the failed path to the cluster head corresponding to the entry has been restored. To sum up, entries referring to nonexistent or unreachable clusters are always eventually evicted.

3.4.3. Routing Address Maintenance

Based on their routing tables, nodes maintain their routing addresses, that is, the hierarchical labels. Label maintenance has to be done throughout the whole network lifetime. When a new node is added to the network, it has to synthesize itself a label that reflects its place in the cluster hierarchy. Likewise, when nodes fail, the cluster hierarchy may need to be modified to guarantee that it can still support hierarchical packet forwarding, notably, that Lemma 3.1 holds. The maintenance activities require updating node labels at various levels, such that the hierarchy reflected in these labels satisfies Properties 1-4.
Listing 3.4: A highly simplified function for refreshing a routing entry. Such a function is invoked to refresh a local routing entry of a node with a routing record received in a heartbeat message from a neighbor. This simple function illustrates only the principal rules of the custom distance-vector algorithm employed by PL-Gossip. It does not consider sequence numbers, time-to-live, and multiple next-hop neighbors, which are crucial in practical implementations of PL-Gossip. The actual function that considers these fields is more intricate, and thus, the reader is referred to the source code (see Appendix C for an overview).

Property [1] always holds. Properties [2-4] in turn, are enforced using local-only operations and gossip-based information propagation, as explained in the remainder of this section.
Responsibility Rule

Let us start with Property 3 which expresses the recursiveness of the cluster hierarchy. It states that two members of the same level-$i$ cluster also belong to the same level-$j$ clusters, for all $j > i$. This implies that, for any two nodes, if the labels of these nodes are equal at level $i$, they have to be equal at all levels $j \geq i$. Maintaining Property 3 thus requires that, for any level-$i$ cluster, any modifications to node labels at levels above $i$ must be performed in a consistent manner by all members of the cluster.

To this end, for label updates in PL-Gossip, a custom single-master update model on a per-cluster basis was developed. More specifically, the head node of a cluster makes all label updates regarding the membership of this cluster in the hierarchy, as formalized by the rule below. Other cluster members simply have to adopt the label updates by such a head.

**Responsibility Rule:** The $(i+1)$-st element of the label of a node can be changed only by the head of the level-$i$ cluster the node is member of. (The identifier of this head is equal to the $i$-th element of the label of the node.)

To facilitate developing intuition behind this rule, the reader should refer to Figure 3.5 on page 71 which visualizes a cluster hierarchy and the corresponding node labels as a tree. The rule states that the head of a cluster is responsible for moving the whole subtree corresponding to that cluster between subtrees corresponding to different superclusters in the label tree. For example, node $R$ in Figure 3.5 would be responsible for moving the subtree corresponding to cluster $C_0$ (the leaf cluster is not visible in the figure) between subtrees corresponding to clusters $C_A^1$, $C_M^1$, and $C_O^1$. Likewise, if the figure had more level-2 clusters, node $A$ would be responsible for moving the subtree corresponding to cluster $C_A^1$ (the light-gray blob below $A$) between those level-2 clusters.

Update Vectors

A label update performed by the head of a cluster is not immediately adopted by all members of the cluster; instead, it propagates gradually via gossiping. This means that for some time after the head has modified its label, the labels of some members of the cluster may be inconsistent. This phenomenon is formalized with two definitions.

Let the *current label* of a node denote the label the node stores locally at a given moment in time, $t$; let us denote it $L_t(A)$ for node $A$. Conversely, let the
**correct label** denote the label the node would store if update propagation were immediate; let us denote it $L^*_t(A)$ for node $A$. Property [1] and the Responsibility Rule determine the relationship between the correct label of a node and the current labels of other nodes as follows. The 0-th element of the **correct label** of a node is equal to the identifier of the head of the level-0 cluster the node belongs to, that is, the identifier of the node itself. The $i+1$-st element ($i \geq 0$) of the **correct label** of the node is equal to the $i+1$-st element of the **current label** of the head identified by the $i$-th element of the **correct label**. This in particular means that the length of the **correct label** is $i+1$ if and only if the head does not have anything in its **current label** at position $i+1$, that is, the length of its **current label** is equal to $i+1$ (it cannot be smaller).

The cluster hierarchy reflected in the correct labels of all nodes satisfies Property [3] as formalized by the following lemma. The proof of the lemma can be found in Appendix A.5.

**Lemma 3.5** For any nodes $A$ and $B$ and any moment in time, $t$, if the correct labels of these nodes are equal at some level $i$, that is, $L^*_t(A)[i] = L^*_t(B)[i]$, then the lengths of these labels are equal, that is, $\text{len}(L^*_t(A)) = \text{len}(L^*_t(B))$, and the labels are themselves equal at all levels higher than $i$, that is, $L^*_t(A)[j] = L^*_t(B)[j]$ for all $i \leq j < \text{len}(L^*_t(A)) = \text{len}(L^*_t(B))$. In other words, when the correct labels are considered, Property [3] holds for all nodes.

The lemma effectively means that the Responsibility Rule guarantees that Property [3] can hold in an actual hierarchy maintained by PL-Gossip, that is, in the labels maintained locally by the nodes. To this end, however, any label updates performed by the head of each cluster have to be eventually adopted by other members of this cluster, which requires appropriate label update propagation mechanisms.

Designing such mechanisms, however, may be challenging. This is because the definition of a correct label involves **global knowledge** about the labels of all nodes in the network. In contrast, in PL-Gossip, a node has only **local knowledge**, that is, it has access only to its own current label and routing table and the current labels and routing tables of its neighbors, as received in their heartbeat messages. In other words, each node has to be able to detect the most recent (freshest) label updates in the state of its neighbors and to apply them to its own label, such that in effect the label will eventually match the correct label. This cannot be done based just on the current neighbor labels. For example, a node with label $P.O.L.I.S.H$ receiving from its neighbor a heartbeat message with label $M.A.L.I.C.E$ is unable to determine whether its label should be modified to $P.O.L.I.C.E$, or whether it should remain unmodified, equal to $P.O.L.I.S.H$. 

---

**Note:** The above text is a continuation of the chapter on enabling hierarchical routing in sensor networks. The text is formatted to match the style of the original document, including the use of italics for emphasis and proper capitalization of terms. The enumeration of properties and rules is consistent with the original text, and the lemma is presented as a formal statement with a specific condition for equality of labels at various levels in the hierarchy.
Therefore, for update propagation, *PL-Gossip* augments heartbeat messages with additional consistency information.

More specifically, it uses a mechanism dubbed *update vectors*. The update vector of a node corresponds to the label of this node and unambiguously specifies the updates applied to the label. The $i$-th element of the vector denotes the sequence number of the last known label update made at level $i+1$ by the head of the level-$i$ cluster the node belongs to, as dictated by the Responsibility Rule. For instance, in Figure 3.10 node $P$ knows that: (1) the last label update performed by $P$ at level 1 has number 1 and wrote $O$ at position 1 of its label; (2) the last update performed by $O$ at level 2 has number 5 and wrote $L$ at position 2 of its label; (3) the last update performed by $L$ at level 3 has number 4 and wrote $I$ at position 3 of its label; (4) the last update performed by $I$ at level 4 has number 5 and wrote $S$ at position 4 of its label; (5) the last update performed by $S$ at level 5 has number 6 and wrote $H$ at position 5 of its label; (6) $H$ acting as the top-level head has not yet made any updates at level 6, that is, $U(P)[5] = 0$ and $L(P)[6] = \phi$.

**Figure 3.10:** A example of a label and a corresponding update vector. Combining the information from its label, $L(P)$, and its update vector, $U(P)$, node $P$ knows that: (1) the last label update performed by $P$ at level 1 has number 1 and wrote $O$ at position 1 of its label; (2) the last update performed by $O$ at level 2 has number 5 and wrote $L$ at position 2 of its label; (3) the last update performed by $L$ at level 3 has number 4 and wrote $I$ at position 3 of its label; (4) the last update performed by $I$ at level 4 has number 5 and wrote $S$ at position 4 of its label; (5) the last update performed by $S$ at level 5 has number 6 and wrote $H$ at position 5 of its label; (6) $H$ acting as the top-level head has not yet made any updates at level 6, that is, $U(P)[5] = 0$ and $L(P)[6] = \phi$.

The update vector of a node is modified whenever the node, acting as a cluster head at some level, changes its label. It is also broadcast in the heartbeat messages of the node, so that other nodes may apply such changes to their own labels. These use-cases are explained in detail below.

**Basic Label Operations**

In total, there are only three operations a node can perform on its label and update vector: *label extension*, *label cut*, and *label combination*. Label extension and label cut are used by a cluster head to change its label. Label combination, in turn, is used to propagate such changes.
Figure 3.11: A example of the label extension operation. By extending its label with $U$, top-level head $V$ makes its cluster, $C_V^i$, a subcluster of a higher-level cluster $C_U^{i+1}$. The operation abides by the Responsibility Rule and modifies the update vector of $V$ accordingly.

Label extension (see Figure 3.11) is executed locally by a top-level cluster head, $V$, when constructing or recovering the hierarchy. By extending its label with $U$, $V$ makes its cluster, $C_V^i$, a subcluster of a higher-level cluster, $C_U^{i+1}$. By extending its label with $V$, in turn, $V$ spawns a new higher-level cluster, $C_V^{i+1}$. Conversely, label cut (see Figure 3.12) is executed locally by a non-top-level head, $V$, when $V$ has detected that its cluster, $C_V^i$, can no longer be a subcluster of a higher-level cluster, $C_W^{i+1}$. This operation removes cluster $C_V^i$ from supercluster $C_W^{i+1}$. Label cut can also be used to dissolve a cluster in order to balance cluster sizes or rotate cluster heads if the set of properties a cluster hierarchy has to satisfy were extended beyond the four properties considered in this chapter. Listing 3.5 presents the pseudo-code for label extension and label cut.

It is crucial to note that both, label extension and label cut, abide by the responsibility rule, that is, they are used by a level-$i$ head to modify the label of the head at level $i+1$. Moreover, in both operations, when modifying its label at level $i+1$, the head, $V$, also writes a new sequence number at the $i$-th position of its update vector (in Figure 3.11 and 3.12, $m \leftarrow$ next update seq. no. of $V$; $U(V)[i] \leftarrow m$). This is consistent with the definition of an update vector and indicates that the label update performed by $V$ is the freshest one, so that other members of $C_V^i$ can also apply the update to their own labels.

Label update adoption (and thereby, propagation) is performed by the third operation, label combination. Label combination is executed whenever a node, $P$,
receives a heartbeat message from a neighbor, $N$ (see Figure 3.13). At that time, $P$ checks for the minimal-level cluster it shares with $N$, by comparing its own local label to the label of $N$, as received in the heartbeat message. More specifically, $P$ looks for the minimal $i$ such that $L(P)[i] = L(N)[i]$. If such $i$ does not exist, then $P$ has just discovered a violation of Property 2 of the cluster hierarchy, which will be propagated through routing tables, as described in Section 3.4.2, and handled by the hierarchy construction algorithm of the head of the top-level cluster of $P$, as explained further in this section.

Otherwise, if such $i$ exists, $P$ determines which of the two labels is fresher at position $i+1$, that is, which of the two nodes has fresher information on the membership of their common level-$i$ cluster in the cluster hierarchy. To this end, $P$ compares its update vector, $U(P)$, with the update vector of $N$, $U(N)$, at position $i$. If $U(P)[i] \geq U(N)[i]$, the label of $N$ is not fresher at position $i+1$ than the label of $P$, and thus, $P$ does not have to adopt any label updates at position $i+1$. Therefore, it continues the whole process by searching for the next $i$ such that $L(P)[i] = L(N)[i]$.

However, if $U(P)[i] < U(N)[i]$, then, starting from position $i+1$, the label of $N$ is fresher than the one of $P$; consequently, $P$ has to adopt the fresh label updates. To this end, $P$ copies the update vector of $N$ at position $i$ ($U(P)[i] \leftarrow U(N)[i]$) and the label of $N$ at position $i+1$ ($U(P)[i+1] \leftarrow U(N)[i+1]$). The copying continues to another position at which the old label of $P$ is equal to the label of $N$. When such
a position is reached, the update vectors of P and N are compared again, and the whole process is repeated, depending which of the two nodes has fresher update information at that position. The pseudo-code of the label combination operation is presented in Listing 3.6.

As all the three label operations abide by the Responsibility Rule, they ensure that the head of a level-i cluster always has the freshest information on the membership of this cluster in the hierarchy. Moreover, by constantly gossiping with their neighbors, nodes gradually propagate any label updates in their clusters. Therefore, the information on the membership of a cluster in the cluster hierarchy gradually becomes consistent for all nodes in the cluster. In effect, the current labels of all nodes become consistent with the correct labels the nodes should have, as formalized by the lemma below. The proof of the lemma can be found in Appendix A.6. Note that while for the sake of simplicity the lemma is formulated for a hierarchy described by Properties 1–4, it can be modified to apply to other cluster hierarchies that support hierarchical routing. This means that the update propagation mechanisms PL-Gossip introduces can be used more broadly, which is demonstrated further in the next chapter.
3.4. ROUTING STATE MAINTENANCE

Figure 3.13: A example of the label combination operation. Upon receiving a label from node N, node P checks if the label contains any new updates that it could apply to its own label. In the two labels, P searches for the clusters it shares with N; for each such cluster, it compares the two update vectors. At position \( i = 1 \), the nodes belong to the same cluster, but P has fresher information on the membership of this cluster in the hierarchy; thus, it does nothing. At position \( i = 3 \), the nodes belong again to the same cluster, but this time N has fresher information. Therefore, P copies from N the fresher part of the label and the update vector, thereby obtaining fresher information on the membership of cluster \( C_i \) in the hierarchy. Likewise, when N invokes the operation on the state received from P, its label will change to N.O.L.I.C.E. It does not matter which one, N or P, applies the updates first.

Lemma 3.6 Consider a quiescent network, that is, a network in which, starting from some time \( t_q \), there are no changes in the connectivity graph reflected in the neighbor relation and no label extension or label cut operations are executed by any node. In such a network, if the cluster hierarchy reflected in the correct labels of all nodes satisfies Properties 1–4, then there exists some time \( t_c \geq t_q \), such that for any node A and any time \( t \geq t_c \), \( L_t(A) = L_{t_q}(A) \). In other words, in a quiescent network the current labels maintained locally by the nodes eventually become consistent with the correct labels the nodes should have.

Since, from the above lemma, the current labels gradually become consistent with the correct labels and, from Lemma 3.5, the correct labels satisfy Property 3, the introduced update propagation mechanisms ensure that the cluster hierarchy eventually satisfies Property 3. Moreover, the requirement for the network quiescence and the weak consistency model — eventual consistency — have been introduced in Lemma 3.6 only to simplify the proof of that lemma. In practice, the mechanisms ensuring the consistency of node labels are highly robust against various failures, including message loss, node failures, and connectivity changes, and some bounds can be given on the time they require to make the node labels consistent after a disruption in the network. The mechanisms can also recover after massive failures in the network, such as partitioning or node isolation, provided that at some point the partitions regain connectivity with the rest of
function combineAddr( 
neighAddr : RoutingAddr, neighUvec : UpdateVector); 
var 
i : integer; 
copying : boolean; 
headID : integer; 
begin 
i ← 0; 
copying ← false; 
while i < min(this.addr.len, neighAddr.len) do begin 
headID ← this.addr[i]; 
if copying then begin 
this.addr[i] ← neighAddr[i]; 
this.uvec[i - 1] ← neighUvec[i - 1]; 
end; 
if headID = neighAddr[i] then begin 
if this.uvec[i] < neighUvec[i] then copying ← true 
else copying ← false; 
end; 
i ← i + 1; 
end; 
if copying then begin 
this.uvec[i - 1] ← neighUvec[i - 1]; 
this.addr.len ← neighAddr.len; 
this.uvec.len ← this.addr.len; 
while i < this.addr.len do begin 
this.addr[i] ← neighAddr[i]; 
this.uvec[i] ← neighUvec[i]; 
i ← i + 1; 
end; 
end; 
end; 
Listing 3.6: The label combine function. It is executed whenever a node receives a heartbeat message from a neighbor. The node determines whether the label of the neighbor contains fresher information on the membership of some clusters in the hierarchy. If so, this information is copied to the local label and update vector of the node. Label combining is thus the mechanism for propagating label updates for a cluster among the members of the cluster.

Given the two operations for modifying a label and the third one that implements a robust mechanism for propagating such modifications, let us see how to construct and maintain the remaining properties of the cluster hierarchy, that is, Property 2 and 4. In fact, hierarchy construction and recovery is performed by detecting violations of Property 2 and 4 in each gossiping round and by reacting to such violations. Nodes learn about the violations from their routing tables. They react to the detected violations by autonomously extending or cutting their labels. This, combined with the above update propagation algorithm,
guarantees that all hierarchy properties eventually hold.

**Hierarchy Construction**

Initially, each node is a top-level head (of its level-0 cluster — Property 1), that is, its label length is equal to one. Hierarchy construction is performed by top-level heads detecting that they are not the sole top-level heads (Property 2 is violated) and reacting to such violations by extending their labels, which corresponds to spawning a new higher-level cluster or joining a cluster to an existing higher-level cluster (see Figure 3.11). The extensions are performed such that the subclusters of the resulting higher-level clusters also satisfy Property 4. Gradually, label extensions eliminate all violations of Property 2, leading to a correct hierarchy: eventually, only a single top-level cluster exists.

The head, $V$, of a top-level cluster, $C_V$, discovers a violation of Property 2 if and only if its routing table contains entries for an adjacent cluster, $C_U$, where $k \geq i$. There are two possible scenarios: first, if $k = i+1$, $V$ can try to make $C_V$ a subcluster of $C_U$; or second, $V$ can spawn a new supercluster, $C_V^i$, hoping that other adjacent level-$i$ clusters will join this cluster, or that it will be possible to make $C_V^i$ a subcluster of some level-$i+2$ cluster. Making $C_V$ a subcluster of $C_U$ corresponds to $V$ extending its label with $U$ at level-$i+1$ (see Figure 3.11). Other members of $C_V$ will gradually learn about the membership update and will extend their labels, as guaranteed by the above update propagation mechanisms. These mechanisms also guarantee that if $C_U^i$ is itself a member of some $C_W^{i+2}$, all members of $C_U^i$ (in particular, the members of $C_V$) will also gradually extend their labels at level $i+2$ with $W$, and so forth. Likewise, spawning a new supercluster, $C_V^{i+1}$, corresponds to $V$ extending its label with $V$.

Making $C_V$ a subcluster of an existing cluster, $C_U^i$, is typically preferred, as it reduces the number of clusters at level $i+1$ as compared to level $i$. However, due to the need for also ensuring Property 4, it is possible only if $C_V$ is adjacent to the central subcluster of $C_U^i$, that is, to cluster $C_U$. Formally, to ensure that the resulting hierarchy satisfies Property 4, $V$ can extend its label at level $i+1$ with $U$ if its routing table contains entries for adjacent $C_U$ and $C_{U}^{i+1}$.

Otherwise, $V$ cannot immediately make $C_V$ a subcluster of any level-$i+1$ cluster; thus, it has to potentially spawn a new level-$i+1$ cluster, $C_V^{i+1}$. However, if all clusters became superclusters at the same time, Property 2 could never be satisfied. In particular, in the beginning, each node forms a single level 0 cluster, so allowing all nodes to create singleton level-1 clusters would not guarantee a correct hierarchy. To cope with this problem, $V$ probabilistically defers spawning a supercluster for a number of rounds. Although different probabilistic heuristics
are possible, for illustrative purposes in this chapter, the following simple one has been adopted.

Upon discovering that it must potentially spawn a supercluster, \( V \) first groups its gossiping rounds into \( S \) virtual slots, each lasting \( r \) rounds. It then randomly selects a slot, \( s \in \{0 \ldots S-1\} \). Finally, it defers spawning a supercluster for \( r \cdot s+1 \) rounds, hoping that in that time some adjacent cluster spawns a supercluster, so that it will be possible to make \( \mathcal{C}_V \) a subcluster of this supercluster.

Using \( S = 2 \) already ensures that the number of clusters at consecutive levels drops exponentially fast, provided that the slot size, \( r \), is long enough. Such a decrement is a direct consequence of the following lemma. The proof of the lemma can be found in Appendix A.7.

**Lemma 3.7** Assume that the slot size, \( r \), is longer than the number of rounds it takes to propagate information between the heads, \( V \) and \( U \), of two adjacent “top-level” clusters \( \mathcal{C}_V \) and \( \mathcal{C}_U \). In this case, with probability \( \geq \frac{1}{4} \), \( \mathcal{C}_V \) will be able to join \( \mathcal{C}_U^{i+1} \) or vice versa.

Oversimplifying things for illustrative purposes, assuming \( S = 2 \) and \( r \) meeting the above assumption, one could expect that roughly half of the clusters (the ones that choose slot 1) will be able to join the superclusters formed by the other half (the ones that choose slot 0), that is, the number of clusters drops exponentially with the level. Not only does this guarantee a correct hierarchy with a single top-level cluster, but also results in a polylogarithmic height of the hierarchy, \( \mathcal{H}^i+1 \), with respect to the total node population size.

A node can choose the slot size, \( r \), satisfying the above requirements based on the entries in its routing table. For example, assuming no message loss, a level-\( i \) head deferring supercluster creation can choose \( r \) equal to the number of hops to the furthest adjacent level-\( i \) head. The nodes may also be more conservative and always set \( r \) to the value computed from Lemma 3.3, which constitutes the upper bound on the distance between two adjacent cluster heads, and consequently, the upper bound on the number of gossiping rounds necessary to propagate information between these heads, assuming a lack of message loss.

Pseudo-code of the hierarchy construction mechanisms described above is presented in Listing 3.7.

**Hierarchy Recovery**

Apart from constructing the hierarchy, PL-Gossip is also able to recover it after failures. To illustrate how the failure recovery works, let us first distinguish two classes of failures. A *benign failure* of a node or a link does not require...
Listing 3.7: The function for synthesizing the hierarchy. It is executed by each node, acting as a “top-level” head, at the beginning of every gossiping round. Its goal is to drive the hierarchy into a state in which only a single top-level cluster exists. To this end, the node first tries to join its cluster to some other higher-level cluster. Otherwise, it checks whether any higher level cluster is necessary at all, that is, if there are other clusters at the same or higher levels. If there are such clusters (but the node cannot join them), it starts the promotion process by drawing a random time slot and waiting till this slot (the suppression counter). When the promotion counter expires the node promotes itself creating a higher level cluster.

any repair apart from removing some routing table entries. Consequently, it is handled by the mechanisms for aging and cleaning node routing tables, described in Section 3.4.2; no modifications to node labels are necessary. In contrast, a disruptive failure, like a higher-level cluster head crash, violates one of the hierarchy properties, namely Property 4. Therefore, it requires repairing the cluster hierarchy by modifying the labels of some nodes. Such repair involves

    function repairPossibleHierIncompleteness();
    var
        i, r, s : integer;
        superClusterCand : RoutingEntry;
    begin
        i ← nodeLevelAsHead(this.addr);
        if i + 1 = this.addr.len then begin
            superClusterCand ← findSuperClusterCandidate(i + 1);
            if superClusterCand ≠ null then begin
                extendAddr(superClusterCand.headID);
                this.suppressionCnt ← -1;
            end else begin
                if this.suppressionCnt > 0 then
                    this.suppressionCnt ← this.suppressionCnt - 1;
                else begin
                    if areEntriesInRoutingTableAtLevelAndAbove(i) then begin
                        if this.suppressionCnt = 0 then begin
                            extendAddr(this.addr[0]);
                            this.suppressionCnt ← -1;
                        end else begin
                            r ← intpow(3, i);
                            s ← random() % NUM_PROMOTION_SLOTS_S;
                            this.suppressionCnt ← r * s;
                        end;
                    end else
                        this.suppressionCnt ← -1;
                end;
            end;
        end;
    end; { function repairPossibleHierIncompleteness }

    function areEntriesInRoutingTableAtLevelAndAbove(
        level : integer ) : boolean;
a head node that detected the failure cutting its label down to the level at which
the failure occurred, which corresponds to removing a subcluster from a no
longer existing cluster (see Figure 3.12). Later if necessary, the above hierarchy
construction algorithm will join such a removed subcluster to a different cluster,
thereby restoring all hierarchy properties.

The head, $V$, of a cluster, $C_V$, which is a subcluster of $C_{i+1}^W$, discovers a
violation of Property 4 if its routing table does not contain an entry for the
supercluster, $C_{i+1}^W$, or an entry for the central subcluster of the supercluster, $C_i^W$, or
such entries do exist, but their adjacency bits are not set. This implies that
$C_V$ should no longer be a subcluster of $C_{i+1}^W$. Therefore, $V$ cuts its label down to
position $i$, which corresponds to removing $C_V$ from $C_{i+1}^W$ (see Figure 3.12).

Such an operation of restoring Property 4 may in turn generate a violation of
Property 2. However, this violation will be subsequently handled by the hierarchy
construction algorithm, described above.

In any case, the introduced update propagation mechanism guarantees that all
members of $C_V$ will adopt the decision of $V$ to leave $C_{i+1}^W$ and later, possibly to
join some other level-$i+1$ cluster. In other words, it guarantees restoring all the
hierarchy properties.

Pseudo-code of the hierarchy construction mechanisms described above is
presented in Listing 3.8.

Label cut could also be used in a similar manner for re-balancing cluster sizes,
if the set of hierarchy properties were extended with properties responsible for
such balancing. It could also be used for rotating cluster heads, which in some
applications may help balancing energy consumption of different nodes. These
use-cases, however, require additional mechanisms that would ensure that the
hierarchy does not oscillate and reaches a stable state. Moreover, they require
changing node labels, which disrupts point-to-point routing. Therefore, for the
sake of simplicity, such mechanisms are not studied in this dissertation; such a
study is left for future work.

Summary

In summary, constructing and maintaining node routing state in PL-Gossip —
the cluster hierarchy reflected in the node routing addresses and routing tables —
corresponds to nodes autonomously ensuring Properties 1–4. The mechanisms
introduced by PL-Gossip to ensure these properties are based on two simple
concepts: asynchronous local gossiping and local operations. Gossiping is used
to fill in and maintain node routing tables and to propagate hierarchy membership
information among nodes. Local operations, in turn, are used to change the
3.4. Routing State Maintenance

Listing 3.8: The function for recovering the hierarchy. It is executed by each node at the beginning of every gossiping round. The node, acting as a head at level $i$, checks whether its cluster should still be a subcluster of the supercluster, that is, whether Property 4 holds for the supercluster. If not, the node cuts its label down to level $i$, thereby removing its cluster from the supercluster. Later the cluster can be joined to some other higher-level cluster by the synthesis function. Thus, eventually all hierarchy properties are restored.

```pseudocode
function repairPossibleHierErrors();
var
i : integer;
superCluster : RoutingEntry;
centralCluster : RoutingEntry;
begin
i ← nodeLevelAsHead(this.addr);
if i + 1 < this.addr.len then
  superCluster ← lookupInRoutingTable(i + 1, this.addr[i + 1]);
centralCluster ← lookupInRoutingTable(i, this.addr[i + 1]);
if superCluster = null or centralCluster = null or
  not superCluster.adjBit or
  not centralCluster.adjBit then begin
  cutAddr(i);
end;
end; { function repairPossibleHierErrors }

function nodeLevelAsHead(addr : RoutingAddr) : integer;
var
i : integer;
begin
i ← 1;
while i < addr.len do begin
  if addr[i] ≠ addr[0] then break;
i ← i + 1;
end;
return i - 1;
end; { function nodeLevelAsHead }
```

membership of the nodes in the hierarchy, that is, to maintain their routing addresses. Gossip-based maintenance of routing tables is performed with a custom hierarchical distance-vector algorithm; in addition, route poisoning is used to cope with possible routing cycles in the presence of failures. Address maintenance, in turn, is a more complex problem. To ensure that the addresses of the members of a cluster are consistent, PL-Gossip adopts a single-master update model on a per-cluster basis, and introduces update vectors, a mechanism for efficiently propagating address updates made by a master node among the slave nodes in a cluster. The master node of a cluster is also the head of the cluster and is dynamically elected using probabilistic heuristics, which effectively corresponds
to constructing the hierarchy. The same mechanisms are used for recovering
the hierarchy after changes in the network, such as node failures or internode
connectivity changes. Overall, the mechanisms introduced by PL-Gossip for
synthesizing and maintaining node routing state aim to be robust, self-managed,
self-contained, and efficient.

For the sake of completeness, Listings 3.9, 3.10, and 3.11 present pseudo-code
of all event handlers of the routing state maintenance mechanisms.

```
function initRoutingState();
var myRoutingEntry : RoutingEntry;
begin
  this.addr.len ← 1;
  this.uvec.len ← this.addr.len;
  this.addr[0] ← this.uniqueID;
  this.uvec[0] ← 0;
  this.addrUpdSeqNo ← 0;
  this.suppressionCnt ← -1;
  clearRoutingTable();
  myRoutingEntry ← allocInRoutingTable(0, this.addr[0]);
  myRoutingEntry.pathLen ← 0;
  myRoutingEntry.adjBit ← true;
  myRoutingEntry.nextHop ← [this.meAsNextHop];
end;  // function initRoutingState

function clearRoutingTable();
```

Listing 3.9: The function for initializing node routing state. Each node initializes its label
and update vector according to Property 1 and puts an entry for its own singleton level-0
cluster in its routing table. It also resets all counters and sequence number generators.

### 3.5. LINK QUALITY ESTIMATION

Having discussed hierarchical packet forwarding and routing state
maintenance in PL-Gossip, let us proceed to discuss the last routing abstraction
identified in Section 3.1, that is, link quality estimation. The goal of link
quality estimation is detecting the neighbors of a node and maintaining up-to-date
information about how good the virtual wireless links to these neighbors are, for
instance, in terms of packet reception rate (PRR).
3.5. Link Quality Estimation

3.5.1. Measuring Link Quality

Because of the unpredictable and lossy nature of low-power wireless communication, link quality estimation has been an important research problem in sensornets. Routing over high-quality links only, instead of over arbitrary ones, can typically reduce the number of transmissions a packet requires to be successfully received by a next-hop node, which in effect reduces the routing stretch and improves the end-to-end packet delivery rates. Moreover, building routing paths using only high-quality links can make the paths more stable, which is particularly important for routing techniques such as graph embedding, compact routing, and hierarchical routing, in which the address of a node depends on the path from the node to some other nodes.

There are a few compelling link estimation algorithms \cite{21, 34, 138}. However, because the functionality they provide is largely independent of the other routing abstractions, they can likely be integrated with \textit{PL-Gossip} and used...
function receiveHeartbeatMsg(
    neighLinkAddr : LinkLayerAddr,
    msg : HeartbeatMsg);

var
i : integer;
begin
combineAddr(msg.senderAddr, msg.senderUvec);
i ← 0;
while i < msg.senderRT.len do begin
    refreshRoutingEntry(
        neighLinkAddr, msg.senderAddr, msg.senderRT[i]);
i ← i + 1;
end;
end;
{ function receiveHeartbeatMsg }

Listing 3.11: The function for handling a heartbeat message reception. This function is invoked each time a node receives a heartbeat message from a neighbor. The node first adopts any fresh label updates using the label combine operation, presented in Listing 3.6. It then refreshes its routing entries with the routing records received from the neighbor; the simplified function for refreshing an entry is presented in Listing 3.4.

interchangeably. For the protocol version described in this chapter as well as for the experiments in the whole dissertation, a standard common passive link estimator \[34, 138\], which requires access to the whole traffic passing through a node, has been employed.

For each datagram a node broadcasts, a passive link estimator generates a new sequence number and embeds this number in the link-layer header of the datagram. In this way, the neighbors of the node can count how many datagrams broadcast by the node were lost and how many were received in a time period. Based on this information, they can compute a reverse packet reception rate for this node in that period. They embed the information on the reverse packet reception rates in their datagrams, so that when the node receives those datagrams, it can learn its forward packet reception rate to any of the neighbors. The bidirectional quality of a link is a combination (e.g., a product or a minimum) of the forward and reverse packet reception rates for the link. This value is typically computed over a period of time or over a number of datagrams and aged over longer periods, for example, using an exponentially-weighted moving average.

The estimator can optionally use additional information to compute the quality of a link. For example, it can count the number of unicast packets acknowledged and unacknowledged by a neighbor, and integrate this information into the link quality for the neighbor. This can improve the reaction time to the changes in the internode connectivity \[34\]. Likewise, it can integrate some information from the physical layer on the state of the wireless channel during
3.5. LINK QUALITY ESTIMATION

reception of a packet, such as the received signal strength indicator (RSSI) or the link quality indicator (LQI). This enables to quickly discriminate between links that have the potential to provide high, stable packet reception rates and links that are likely to oscillate [34].

PL-Gossip can employ a passive estimator because it proactively maintains node routing state, thereby generating traffic. Nodes gossip with each other by periodically broadcasting heartbeat messages, and thus, a passive link estimator can count these messages to detect the neighbors of a node and to obtain a base for their link quality estimates. Link estimation requires only augmenting a heartbeat message with a link-estimator header and footer. The footer contains records with the reverse link quality information for the neighbors of the sending node, the header, in turn, the number of such records and a heartbeat sequence number.

3.5.2. Using Link Quality Measurements

PL-Gossip makes use of the computed link quality estimates in a few ways, mainly to improve the robustness of the protocol to packet loss.

First, when building routes, each node considers only those neighbors to which the bidirectional packet loss does not exceed a certain threshold. In this way, the node can minimize the number of transmissions necessary at each routing hop to deliver a packet to the next hop. In addition, a routing path consisting of only high-quality links is typically more stable than a path consisting of arbitrary links; consequently, a hierarchy built on top of high-quality links can be more stable as well.

Second, when forwarding a packet, a node can choose the neighbor with the highest-quality link, which is selected from the list of potential next-hop candidates associated with the routing entry for the target cluster. This can further reduce the number of transmissions necessary to forward the packet to the next hop.

Third, the estimates can be used for extending various timeouts in a highly-conservative version of PL-Gossip. From the minimal quality threshold a link has to exceed to be considered for a routing path, a node can compute the expected packet loss for any path. Consequently, it can extend various timeouts to account for the expected packet loss when propagating information, for example, the duration of the virtual slot size, r, when a node has to potentially spawn a new higher-level cluster, as mentioned in Lemma[3.7].

In general, bidirectional link quality estimates constitute a measure of the expected packet loss of a link or a path. Therefore, PL-Gossip uses such estimates to improve its robustness against this types of packet loss.
In addition, however, \textit{PL-Gossip} employs mechanisms for dealing with temporal packet loss, or \textit{variations} in packet loss. In packet forwarding, if a next-hop node does not acknowledge the reception, a forwarding node chooses a retransmission back-off that is long enough to deal with any potential temporal packet loss \cite{119}. In routing state maintenance, in turn, each next-hop candidate of a routing entry is given an age, which can grow for a few gossiping rounds before the candidate is evicted if it has not been refreshed. Therefore, even when a heartbeat message refreshing the candidate is lost, it is likely that a subsequent message will be successfully received and will refresh the candidate. Preserving such a candidate in the routing table improves the stability of the hierarchy. Overall, these mechanisms for coping with variations in packet loss aim at further improving the robustness of \textit{PL-Gossip}. Any loss they are unable to tolerate is simply treated as a link failure and is handled with standard mechanisms for such a failure, as described in the previous sections.

\section*{3.6. CONCLUDING DISCUSSION}

To sum up, the objective of \textit{PL-Gossip} is to enable hierarchical routing in sensornets, and thereby, to address the problem of providing a small-state, small-stretch, robust, self-managed point-to-point routing protocol for sensornets, as formulated in Section \ref{sec:1.3}. Small node state in \textit{PL-Gossip} is expected to result from the inherent properties of a cluster hierarchy and the heuristics devised for constructing and maintaining such a hierarchy. Small stretch may hopefully be achieved considering the geometric nature of sensornet deployments and the theoretic analyses of the routing stretch in such networks, mentioned in Section \ref{sec:2.3.4}. Robustness is ensured by the mechanisms introduced for maintaining the cluster hierarchy, in particular, their emphasis on locality, and the mechanisms for tolerating and recovering from packet loss. Finally, self-management is an inherent property of the \textit{PL-Gossip} protocol itself. \textit{PL-Gossip} will be evaluated with respect to these properties in the following chapters.

However, by no means does \textit{PL-Gossip} aim to be the ultimate hierarchical routing protocol for sensornets. Instead, the goal was to design \textit{PL-Gossip} as a \textit{basis} for such a protocol; the basis itself would have the potential to work in sensornet deployments, but at the same time would make future improvements possible. Having a basic albeit practical protocol enables evaluating hierarchical routing in the real world. Moreover, once the real-world performance of this routing technique has been studied, the basis could be a good starting point for
developing production-quality solutions.

This, in particular, is the reason why self-management and self-containment were emphasized in the design of \textit{PL-Gossip}. In the experimental phase, they facilitate extensive testing, thereby simplifying protocol evaluation. Similarly, in the production phase, they can simplify integration, and subsequently, deployment and maintenance. However, a production-quality protocol may trade these three for performance. For example, it is desirable for the hierarchy maintenance mechanisms to be self-contained, but if other protocols can provide some feedback to these mechanisms to improve their performance, such feedback may be implemented when the protocol is a part of a system, thereby breaking its self-containment property. Likewise, although self-managed routing address synthesis simplifies deployment and maintenance, in some environments manually-configured addresses may be more suitable, such as in buildings in which the addresses may be required to reflect a room hierarchy. The basis, which \textit{PL-Gossip} is, can be modified like above in a number of places, as is signaled throughout the remaining chapters of this dissertation.
Chapter 4

Evaluating Hierarchical Routing in Sensornets

Since a point-to-point routing protocol for sensornets is required to offer small state, small stretch, robustness, and self-management capabilities, PL-Gossip is evaluated experimentally with respect to these properties. The evaluation is conducted on three common experimental platforms for sensornets: in a custom high-level simulator, which is describe below, in TOSSIM 2.0, a low-level sensor node simulator with realistic low-power wireless models, and on real sensor nodes that comprise the experimental testbed built at Vrije Universiteit Amsterdam. Wherever possible, the experimental findings are supported with relevant theoretical results.

As discussed in Section 2.3.4, to the best of my knowledge, the experiments that involve the actual implementation of PL-Gossip, that is, TOSSIM and testbed experiments, are the first such experiments reported for hierarchical routing in the sensornet literature. More importantly, however, these experiments are not limited to just PL-Gossip. Instead, based on some earlier experience with the
implementation of PL-Gossip and other routing techniques, a hierarchical routing library has been implemented that enables evaluating different design solutions for hierarchical routing, as encountered in other proposed protocols. In particular, using the library the performance differences between landmark and area hierarchies are shown, as well as the differences between various mechanisms proposed for propagating hierarchy information. All in all, this chapter constitutes a comprehensive evaluation of hierarchical routing in sensornets.

The rest of the chapter is organized as follows. Section 4.1 starts with the results from high-level simulation experiments of PL-Gossip, which were conducted in the custom simulator. Section 4.2 gives an overview of the design of the hierarchical routing library and discusses the design decisions that impact the state, stretch, and robustness of hierarchical routing the most, and which are evaluated later. Section 4.3 evaluates the implementation of the library in realistic low-power networks simulated in TOSSIM; in particular, it studies how different design decisions impact the performance of hierarchical routing. Section 4.4 in turn, evaluates the library on the aforementioned sensornet testbed and shows how different real-world phenomena affect hierarchical routing. Finally, Section 4.5 concludes by summarizing the merits and drawbacks of PL-Gossip, the library, and hierarchical routing in general. A guide to the library sources can be found in Appendix C.1 The testbed, in turn, is described in Appendix B

4.1. HIGH-LEVEL SIMULATION

A high-level simulator is typically the first experimental platform for evaluating any routing protocol. High-level simulation eliminates the resource constraints of sensor nodes and abstracts many of the aforementioned peculiarities of low-power wireless communication. As a result, it allows for experimenting with very large networks and for conducting similar experiments multiple times using different configuration parameters and random seeds. Therefore, high-level simulation is typically used to verify whether a protocol does not contain any apparent algorithmic flaws and whether its performance scales as expected, at least in idealized conditions. In particular, all hierarchical routing protocols proposed before PL-Gossip have been evaluated only with high-level simulation.

4.1.1. Simulation Settings

To conduct high-level simulation experiments, a custom simulator has been written. The assumptions regarding the performance of sensor nodes and the
properties of low-power wireless communication that have been made for the simulator are the same as in other similar high-level simulators for sensornets [16, 22, 35, 61, 75, 84, 94, 100, 111]. Firstly, the simulator assumes a unit-disk radio connectivity model: all nodes have the same fixed circular radio range and a node has links only to all the nodes that fall within its range. Secondly, it ignores the capacity of and the congestion in the air: all nodes have unlimited radio throughput, packets can have an arbitrary length, and there are no packet collisions. Thirdly, there is either no packet loss in the whole network or the packet loss is fixed to the same value for all wireless links. These assumptions oversimplify the reality greatly. However, as will be demonstrated in implementation-based experiments, the performance of PL-Gossip does not suffer much in the real world, even though the assumptions are violated.

For conducting high-level simulation experiments, the variant of PL-Gossip described in the previous chapter has been employed. However, due to the simplicity of the low-power wireless communication model used in the simulator, the link estimation functionality was not necessary, and consequently, was excluded from the protocol code. For the same reason, the packet forwarding functionality was simplified to involve just next-hop neighbor selection for a packet. No such modifications were possible for routing state maintenance, and thus, this functionality remained precisely as described in the previous chapter. As routing state maintenance is the most intricate functionality in a hierarchical routing protocol, leaving this functionality in its entirety allowed for studying how the mechanisms introduced by PL-Gossip for hierarchy maintenance can perform in an idealized environment.

Such a variant of PL-Gossip was simulated in a number of configurations and experimental scenarios, such as varying node population sizes (denoted $N$ in this chapter), densities (denoted $\rho$ in this chapter), and placement strategies (denoted $p$ in this chapter). In addition, different traffic patterns and failure scenarios were tested. Since the results were consistent, for the sake of brevity, only their small, albeit illustrative subset is presented in this dissertation.

In those experiments, nodes were placed in a square area and either arranged into a grid or deployed randomly in the area. These two configurations correspond to two extremes in node placement strategies: a grid is highly regular, while random placement is completely irregular. The density of the network, measured in the number of neighbors each node had, was low. In particular, in the grid deployment a node had at least 5 (corner nodes) and at most 12 (most of the nodes) neighbors. All nodes were booted simultaneously in round 0, and were given a warm-up period of several rounds until the hierarchy had been bootstrapped, that is, until they all had equal-length labels with the same last element. The number of
slots, \( S \), used by a head deferring spawning a supercluster was fixed to 10 at level 0 and 2 at higher levels. The higher value at level 0 was to effectively deal with denser networks, which were used in other experiments: less nodes promoted themselves to level-1 cluster heads. When the hierarchy had been bootstrapped, the values of selected performance metrics were measured, routing was initiated, or some other experimental scenarios were run.

### 4.1.2. Routing State

Since hierarchical routing promises a significantly smaller routing state as compared to other techniques, the first metric to be measured for PL-Gossip is routing state. One of the factors that determines the routing state of a node in hierarchical routing is the height of the hierarchy, which in particular corresponds to the length of a routing address. Kleinrock and Kamoun have proved that in hierarchical routing a minimal routing state is achieved for cluster hierarchies the height of which depends logarithmically on the total node population size [66].

Figure 4.1 presents the height of the hierarchies obtained in the considered experiments with PL-Gossip. The hierarchy height appears to depend polylogarithmically on the node population size in both regular and irregular networks. Moreover, the values are rather stable: the standard deviation is small and the 99-th percentile is close to the average. A polylogarithmic hierarchy height is possible in PL-Gossip because of two features of the protocol. First, the maximal distance between two cluster members in the employed sample hierarchy model grows exponentially with the cluster level, as formalized by Lemma 3.4. This means that just logarithmic hierarchy height is sufficient for the top-level cluster to cover all nodes. Second, the probabilistic heuristics PL-Gossip uses for spawning clusters at subsequent levels aim at reducing the number of clusters exponentially with the level, as formalized by Lemma 3.7. This means that, on average, the number of clusters at subsequent levels can actually be reduced to the required one top-level cluster in a polylogarithmic number of steps.

The dominating factor in the routing state of a node, when measured in bytes, is the routing table of the node. Since the size of a routing entry in hierarchical routing is considered constant, the size of a routing table is typically measured in the number of entries the table contains. Kleinrock and Kamoun showed that in a hierarchy with a logarithmic height the number of routing entries can also depend polylogarithmically on the total node population size [66].

Figure 4.2 depicts the number of routing entries per node in PL-Gossip. Like the hierarchy height, the average and the 99-th-percentile number of routing entries appears to depend polylogarithmically on the node population size in
4.1. **HIGH-LEVEL SIMULATION**

**Figure 4.1:** The cluster hierarchy height (high-level simulation). The figure depicts the average (including the standard deviation) and the 99-th-percentile height of the cluster hierarchy for exponentially growing node populations deployed sparsely on a grid (left) or in a random fashion (right). For every network size, the aggregates have been computed over 100 independent simulation runs, each resulting in a different cluster hierarchy. The height of the hierarchy constructed by PL-Gossip appears to depend polylogarithmically on the node population size. This is possible because of the sample properties PL-Gossip introduces for a hierarchy and the probabilistic heuristics it uses to construct and maintain the hierarchy.

Both grid and random networks. Moreover, the values are relatively small. For example, a node in a 4096-node network on average requires less than 40 routing entries and in 99% of the cases does not require more than 55 entries. This is again a consequence of the sample properties of the cluster hierarchy and the probabilistic heuristics used to construct and maintain the hierarchy. Note that the presented variant of PL-Gossip does not introduce any properties or mechanisms for balancing cluster sizes, which in theory are necessary to ensure logarithmic routing tables. Nevertheless, even without these mechanisms, the probabilistic heuristics employed by PL-Gossip produce hierarchies that result in polylogarithmic routing tables.

A polylogarithmic hierarchy height, which reflects the size of a routing address, and a polylogarithmic number of entries, which determines the size of a routing table, illustrate that the routing state of a node in PL-Gossip can be polylogarithmic, at least in static idealized networks. This suggests that PL-Gossip supports hierarchical routing well, and, in general, has the potential to be a small-state point-to-point routing protocol for sensornets.
Figure 4.2: The routing table size (high-level simulation). The figure depicts the average (including the standard deviation) and the 99-th-percentile number of routing entries per node for exponentially growing node populations deployed sparsely on a grid (left) or in a random fashion (right). For every network size, the aggregates have been computed over 100 independent simulation runs, each resulting in a different cluster hierarchy. For example, an aggregate for a 4096-node network was computed for \(4096 \times 100\) values. The number of routing entries in PL-Gossip appears to depend polylogarithmically on the node population size. Like in the case of hierarchy height, this is possible because of the sample properties the presented variant of PL-Gossip introduces for a hierarchy and the probabilistic heuristics it uses to construct and maintain the hierarchy.

4.1.3. Routing Stretch

As theoretical work on routing suggests, a reduced routing state increases routing stretch above the optimal value of 1 \([70]\). Since hierarchical routing can reduce its state substantially, such a reduction in state can result in a large increase of stretch. For example, in a network with full connectivity, a stretch of hierarchical routing may grow as \(O(\log N)\). Similarly, in small-world topologies, such as the current Internet, the stretch of hierarchical routing may be large. To be practical for sensornets, however, PL-Gossip has to deliver a stretch that is close to 1 on average and is also small in a great majority of the cases.

Figure 4.3 presents the routing stretch of PL-Gossip as obtained in the considered high-level simulation experiments. The figure illustrates that the stretch offered by PL-Gossip is relatively small and beyond certain point grows extremely slowly with the node population size, at least in the considered unit-disk networks. In particular, the average stretch in both grid and random graphs does not exceed 1.5. Due to more irregularities in the internode connectivity of random networks as compared to grids, the average stretch and its variance are larger in random networks than in grids. Overall, however, there is little difference between the stretch of different nodes, such that the stretch
is close to 1 on average and in 99% of the paths it does not exceed 3 and 4, respectively, for grid and random networks.

Figure 4.3: The routing stretch (high-level simulation). The figure depicts the average (including the standard deviation) and the 99-th-percentile routing stretch in the networks from Figure 4.1 and 4.2. Since in those high-level simulation experiments there was no message loss and no congestion, the routing stretch was measured like in wired networks — in terms of hop stretch — the ratio between the number of hops on a route between two nodes and the number of hops on the shortest possible path between those nodes. For every network size, the aggregate hop stretch was computed over all node pairs and 100 different cluster hierarchies. For example, an aggregate for a 4096-node network was computed for 4096 · 4095 · 100 paths. On average, the hop stretch offered by PL-Gossip is relatively close to the minimal value of 1, which means that the routing paths PL-Gossip selects are close to the optimal ones. The 99-th percentile hop stretch is also relatively small. These results are possible because sensornets have “geometric” topologies. In such topologies, the state aggregation caused by hierarchical clustering does not entail a large increase in stretch.

The above results imply that, despite maintaining a small routing state, PL-Gossip can provide a stretch that is close to the optimal one in sensornets. While the results may initially seem to contradict the theory on hierarchical routing, on closer examination they are perfectly valid. Hierarchical routing does result in a large routing stretch in some topologies, such as cliques or small-world networks. The topologies formed by sensornets, however, fall into a different category. Because of their embedding in physical space and their limited radio range, sensor nodes typically form topologies with diameters that are relatively large compared to the network size. More specifically, the diameters of sensornets grow as $N^\nu$ with the node population size (where $0 < \nu \leq 1$). For example, $\nu$ can be close to $1/2$ in planar (2-dimensional) networks and close to $1/3$ in volumetric (3-dimensional) networks. The theory shows that, in such “geometric” topologies, the aggregation of routing information caused
by hierarchical clustering does not entail a large increase in stretch \[66, 70\]. In other words, in typical topologies of sensornets, hierarchical routing, in general, and PL-Gossip, in particular, can simultaneously offer a small stretch and a small state. This property makes PL-Gossip an attractive point-to-point routing protocol for sensornets.

### 4.1.4. Robustness and Self-Management

Hitherto, it has been demonstrated that, in static networks, PL-Gossip can simultaneously provide small routing state and small routing stretch. To complete the high-level simulation experiments, the subsequent results illustrate how the protocol works in the presence of network dynamics, notably how it bootstraps the cluster hierarchy and recovers it from changes in the network topology.

#### Hierarchy Construction

Figure 4.4 depicts the number of gossiping rounds that are necessary to bootstrap a complete cluster hierarchy. In those experiments, all nodes were started simultaneously and the hierarchy was considered as bootstrapped when all the nodes had equal-length labels with the same last element, that is, when just a single top-level cluster existed. Because the information on various-level clusters has to propagate throughout the network to enable spawning higher-level clusters, the bootstrap latency depends on the diameter of the network, which is also visible in the depicted experimental results: the number of bootstrap rounds grows exponentially with the exponentially growing network diameter.

Nevertheless, the absolute values indicate that hierarchy bootstrap in PL-Gossip is relatively fast as compared to the network diameter. For instance, for a 1024-node grid network with a diameter of 32 hops, the hierarchy is formed within 41.74 rounds on average and at most 87 rounds in 99% of the cases. This is because the introduced local-only mechanisms for modifying the hierarchy and propagating the modifications enable PL-Gossip to construct the hierarchy concurrently at multiple levels.

Moreover, simultaneous boot is the worst-case scenario: there are no higher-level clusters formed, and, consequently, all nodes have to potentially spawn such clusters. Normally, in real-world deployments, networks are built incrementally by adding one node after another. In such a scenario, PL-Gossip typically finishes the bootstrap process within a few rounds after the last node has been added (results not plotted). This is because adding a node to a network typically does not require spawning any new higher-level clusters, as the node can usually join some already existing level-1 cluster. The probability that this is not
4.1. High-Level Simulation

Figure 4.4: The hierarchy bootstrap latency (high-level simulation). The figure depicts the average (including the standard deviation) and the 99-th-percentile number of gossip rounds necessary for PL-Gossip to fully bootstrap a cluster hierarchy in exponentially growing node populations deployed sparsely on a grid (left) or in a random fashion (right). For every network size, the aggregates have been computed over 100 independent simulation runs, each resulting in a different cluster hierarchy. The number of bootstrap rounds depends on the diameter of the network, and thus, it grows exponentially with the exponentially growing network size. Nevertheless, it is relatively fast as compared to the network diameter. Moreover, simultaneous boot, as in the figure, is the worst-case scenario. In a more typical incremental boot, PL-Gossip performs much better, requiring on average a constant number of rounds to admit a new node to the network.

To further demonstrate that the hierarchy construction mechanisms introduced by PL-Gossip perform well, PL-Gossip is compared with an alternative approach. A straightforward alternative for bootstrapping a cluster hierarchy is a centralized approach. Firstly, all nodes discover their neighbors and build a spanning tree rooted at a special gateway node. Secondly, each node reports its neighbors to the gateway by routing up the spanning tree over a multihop shortest path. Thirdly, based on the complete connectivity information, the gateway computes the cluster hierarchy, that is, it precomputes the routing address and the routing table for each node. Finally, the precomputed routing state for each node is routed from the gateway to that node, again along the shortest path down the tree.

Figure 4.5 compares the total number of messages necessary for bootstrapping a cluster hierarchy in an optimal centralized approach (excluding the cost of building the tree and discovering node neighbors) with the corresponding number for PL-Gossip. The figure illustrates that the message
cost of PL-Gossip is close to that of the optimal centralized approach, and the larger the network, the smaller the difference between the two approaches. In particular, for a 4096-node network, the total number of messages required by PL-Gossip to bootstrap a cluster hierarchy is less than twice the number of messages required by an optimal centralized approach. Moreover, while in PL-Gossip each node contributes equally to the bootstrap message cost, in the optimal centralized approach the cost is highly imbalanced. For example, in the 4096-node experiments in the figure, the five nodes around the gateway transmitted in total 8187 messages, whereas many perimeter nodes transmitted only one message each. Such a high traffic imbalance of the centralized approach results in a nonuniform resource consumption, which may be problematic in some scenarios. Finally, while the centralized approach performs well in hierarchy construction, it may be too expensive for failure recovery, especially since the recovery not only involves the cluster hierarchy, but also the spanning tree used to maintain the hierarchy. In contrast, in terms of the number of messages, the traffic of PL-Gossip does not change in the presence of failures, as will be demonstrated shortly. In addition, the mechanisms PL-Gossip introduces for hierarchy bootstrapping will be compared with other alternative mechanisms further in this chapter.

Figure 4.5: Hierarchy bootstrap in PL-Gossip vs. an optimal centralized approach. The figure depicts the average total number of messages necessary for PL-Gossip and an optimal centralized approach to fully bootstrap a cluster hierarchy in exponentially growing node populations deployed sparsely on a grid. For every network size, the aggregates have been computed over 100 independent simulation runs, each resulting in a different cluster hierarchy. The message cost of PL-Gossip is close to that of the optimal centralized approach, and the larger the network, the smaller the difference between the two approaches. Moreover, PL-Gossip outperforms the centralized approach when other metrics, such as traffic balance, are considered (not plotted).
Overall, in terms of the latency of hierarchy construction, PL-Gossip performs relatively well in both simultaneous network bootstraps and incremental network deployments. Moreover, in terms of the number of messages necessary to construct the hierarchy, PL-Gossip performs comparably to an optimal centralized approach, and even outperforms that approach when other metrics, such as traffic balance, are considered.

Hierarchy Maintenance

Hierarchy construction, however, is just the beginning of a more general maintenance process, which has to be performed throughout the whole network lifetime. In practice, the topology of a network is not static: it changes due to changes in the internode connectivity or in the node population. Such changes may sometimes violate some of the hierarchy properties. In effect, PL-Gossip may be forced to repair the hierarchy to restore the violated properties.

To illustrate how the mechanisms for hierarchy maintenance introduced by PL-Gossip perform in high-level simulation, the following experiment has been chosen, the results of which are depicted in Figure 4.6. In that experiment, a 1024-node network was operating for 21,000 rounds. In any round, some 128 nodes out of 1024 (12.5%) were dead, while the remaining 896 nodes were alive. In the initial 1000 rounds, there were no changes in the node population. During the next 10,000 rounds, node churn of a given rate was applied. In particular, with a churn rate of 4, in every round 2 random live nodes were killed and 2 random dead nodes were restarted. Afterward, during the last 10,000 rounds there was again no churn. In addition, during the whole experiment, each link exhibited 10% packet loss. Finally, to amplify the performance degradation of routing under failures, each routing entry could have only one next-hop candidate, which meant that when the candidate died, the whole entry became invalid.

32 randomly selected nodes were always alive and were used for measuring the routing quality by letting them send messages to each other. In particular, Figure 4.6 depicts the evolution of the internode reachability (top), that is, the existence of routing paths between the selected 32 \cdot 31 = 992 node pairs, the hop stretch of such paths (middle), and the average size of the routing tables over all 896 live nodes (bottom).

While the experiment is largely unrealistic, it illustrates well how various phenomena can affect the performance of PL-Gossip. Due to repeated message loss triggering hierarchy changes, the reachability (top plot) occasionally falls during the initial 1000 and the last 10,000 rounds. This is because with the given message loss rate it is likely that some node falsely determines that
CHAPTER 4. EVALUATING HIERARCHICAL ROUTING IN SENSONETS

Figure 4.6: PL-Gossip under network dynamics (high-level simulation). The figure depicts the evolution of the internode reachability (top), hop stretch (middle), and routing table sizes (bottom) under sample network dynamics, notably node churn and packet loss, in a sparse 1024-node grid network. In general, the figure illustrates that PL-Gossip is robust to network dynamics: nodes running the protocol can autonomously handle the dynamics, potentially with some performance degradation.

$N$: 1024; $\rho$: sparse; $p$: grid

a link has failed. If such a “failure” triggers a membership change for a cluster, the communication to and from the cluster is temporarily disrupted (the communication within the cluster is preserved). This reduces reachability for a number of nodes, depending on the level of the cluster in the hierarchy. Node churn, which introduces real failures in the network, amplifies this effect causing greater oscillations in reachability.

Similarly, network dynamics generate peaks in the hop stretch (middle plot). This is because propagating a new short route via a just-booted node requires some time.

Node churn also leads to larger routing tables (bottom plot). It takes a few rounds, depending on the time-to-live of a routing table entry, to determine that a node is dead or a cluster ceased to exist. Therefore, the routing tables can be polluted with entries for no longer existing clusters. In addition, new nodes
are constantly added to the system, further increasing the node routing tables. Nevertheless, even under high churn, the average routing table size is relatively small and stable, and it decreases fast when the churn stops.

Overall, the experiment illustrates that nodes running PL-Gossip can autonomously handle network dynamics. Whenever some nodes fail or the internode connectivity changes, the affected nodes detect the change in the network topology and account for the change to restore any violated hierarchy properties. This indicates that, in addition to providing small state and small stretch, PL-Gossip can be a robust, self-managed routing protocol for sensornets. More robustness-related results are presented further in this chapter.

4.2. IMPLEMENTING HIERARCHICAL ROUTING

Although high-level simulation is invaluable for studying the algorithmic aspects of a protocol in idealized environments, it does not capture many phenomena that occur in the real world. Real-world protocol deployments to date have demonstrated that especially in sensornets practice very often diverges from theory. The severe resource constraints of sensor nodes and the peculiarities of low-power wireless communication often make the proposed protocols fail to deliver the expected performance in the real world or, even worse, they make implementation of such protocols intractable for sensor nodes. For this reason, demonstrating the performance of a sensornet protocol in practice requires experimentally evaluating an actual implementation of the protocol, typically using both low-level simulation and actual sensor node hardware.

To this end, a hierarchical routing library has been developed in TinyOS 2.0 [88], an operating system for sensornets. More specifically, in the course of the presented research, several different variants of PL-Gossip have been implemented, also including some appealing solutions proposed in competing protocols. That experience led to a hierarchical routing framework that captures some selected common characteristics of many proposed hierarchical routing protocols and at the same time identifies various design points that differentiate the protocols and allow for exploring the design space. Implementing the framework with different solutions at those design points provided a means to systematically evaluate and compare in the real world many mechanisms proposed for hierarchical routing, as well as to test some novel ideas. To the best of my knowledge, the hierarchical routing library is the first implementation of hierarchical routing for sensornets reported in the literature.

The remainder of this section, first, gives an overview of some major design
CHAPTER 4. EVALUATING HIERARCHICAL ROUTING IN SENSORNETS

points and the protocol scheme in the library. Then, it focuses on two major design points, which essentially determine routing state, stretch, and robustness, but have not been studied thoroughly to date. Finally, it give some remarks regarding the implementation of hierarchical routing for resource constrained sensor nodes.

4.2.1. Sample Design Points

The first major design point is the type of the cluster hierarchy. There are two classes of hierarchies: area hierarchies [66] and landmark hierarchies [131]. An area hierarchy, described in the previous chapter, is created by logically grouping nodes into areas, grouping areas into superareas, and so on. A landmark hierarchy, described in detail in Section 4.2.3 is in turn created by appointing some nodes as various-level landmarks and binding other nodes to the closest landmark at each level. There are subtle differences and trade-offs in the hierarchical routing algorithms for these two different hierarchy types and the performance of these algorithms, as explained analytically by Tsuchiya [131] and demonstrated experimentally further in this chapter. Although in the previous chapter a variant of PL-Gossip using an area hierarchy was described, the protocol was designed so that it could work seamlessly also with a landmark hierarchy.

Each of the hierarchy types introduces its own design points, one prominent design point being recursiveness. From its definition, an area hierarchy is inherently recursive, that is, in an area hierarchy, each level-i cluster is completely nested in some level-i+1 cluster (cf. Property 3 in Section 3.2.1). In contrast, although a landmark hierarchy can be recursive, which requires that two nodes bound to the same level-i landmark should also be bound to the same level-i+1 landmark, it does not have to [16, 75, 131]. Maintaining a nonrecursive hierarchy is less intricate than a recursive one because the maintenance mechanisms for a nonrecursive hierarchy do not have to ensure that nodes with labels equal at level i also have their labels equal at all levels j ≥ i, which may be challenging (recall the Responsibility Rule and update vectors introduced in Section 3.4.3). However, a recursive hierarchy enables more efficient, per-cluster notifications of routing address changes as compared to per-node notifications in a nonrecursive hierarchy. For this reason, while PL-Gossip can support both recursive and nonrecursive hierarchies, the hierarchical routing library focuses on recursive ones.

Another design point for each of the hierarchy types are the actual properties of a hierarchy. The hierarchy properties can affect cluster balancing and scaling at subsequent levels, which in turn allows for exploring the state-stretch trade-off
4.2. Implementing Hierarchical Routing

within hierarchical routing itself. For an area hierarchy, the previous chapter introduced the four properties, which enable scaling cluster diameters at each subsequent level \(i\) as \(O(3^i)\). However, different sets of properties for an area hierarchy can be devised easily. Likewise, for a landmark hierarchy, a common cluster scaling function is \(2^i\), or, more generally, \(\alpha^i \cdot \alpha_0\), where \(\alpha > 1\). Again, different functions or additional constraints on clusters at subsequent levels are possible. Since the properties of a cluster hierarchy for PL-Gossip can be varied to a large extent, it will be demonstrated shortly how different hierarchies in PL-Gossip allow for exploring the state-stretch trade-off within hierarchical routing.

Apart from such design points regarding a cluster hierarchy itself, there are other design points associated with the manner the hierarchy can be maintained. To begin with, node routing addresses (labels) can be synthesized offline, prior to a network deployment [42, 131], or at runtime, using some self-managed mechanisms like in the variant of PL-Gossip presented in the previous chapter. In some deployments, pre-configured addresses may be more suitable. For example, in indoor sensor networks they may reflect the hierarchy of building space, that is, rooms within corridors within floors within wings and so on, such that in effect not only do they serve for addressing nodes, but also, for instance, for labeling and aggregating data produced by the nodes [56, 75]. However, pre-computing node addresses may entail a high deployment cost, especially since low-power wireless connectivity is highly unpredictable, and thus, after the deployment it may turn out that a pre-constructed hierarchy is invalid because nearby nodes, which have been expected to communicate, cannot hear each other. In addition, reconstructing node addresses offline after a cluster head failure may be inefficient in large networks. Therefore, in the hierarchical routing library, self-managed address synthesis mechanisms, like in PL-Gossip, have been implemented.

Due to the aforementioned complexity of the hierarchy construction problem, self-managed address synthesis is performed using heuristic mechanisms. One design point for such mechanisms is the order in which clusters at subsequent levels are created: top-down or bottom-up. In a top-down approach, the clusters are constructed from the top level down the hierarchy [126]. In an area hierarchy, a single top-level supercluster is split into multiple lower-level clusters, each of which is again split into subclusters, and so forth, down to level 0 with the granularity of a single node. Likewise, in a landmark hierarchy, a few nodes are appointed as top-level landmarks, each of which subsequently appoints lower-level landmarks, and so on down the hierarchy. In contrast, in a bottom-up approach (like PL-Gossip), the clusters are constructed from level 0 up the hierarchy. In an area hierarchy, singleton level-0 clusters are merged into level-1
clusters, which are in turn merged into level-2 clusters, and so forth. In a landmark hierarchy, in turn, some of the nodes are promoted to level-0 landmarks, some of which are further promoted to level-1 landmarks, and so on. In PL-Gossip and the hierarchical routing library, bottom-up mechanisms have been used, as top-down ones are considered to have problems adapting to some network topologies, such as topologies with nonuniform node densities [75]. Nevertheless, investigating top-down approaches to hierarchy construction in sensornets may constitute an intriguing avenue for future work.

Heuristic hierarchy maintenance mechanisms can further be divided into deterministic [42] and probabilistic [5, 122]. Deterministic heuristics typically give some guarantees about the extent to which the resulting hierarchy differs from an optimal one, for example, in terms of the routing state at each node. Probabilistic heuristics, like in PL-Gossip, rarely give any guarantees, but strive to ensure that with a high probability, the resulting hierarchy will yield reasonable performance of hierarchical routing. Deterministic heuristics typically involve multiple steps and often require global knowledge about the internode connectivity; consequently, they can be prohibitively expensive in sensornets. For this reason, in the hierarchical routing library, like in PL-Gossip, probabilistic hierarchy maintenance heuristics have been adopted.

For the heuristics to work, nodes have to be able to propagate up-to-date information about the clusters throughout the network. This is closely related to maintaining the routing tables of the nodes and constitutes an important, albeit hardly studied design point, which impacts the robustness of hierarchical routing. There are two major approaches to such information propagation: multi-level scoped-flooding [5, 22, 75, 122] and local-only gossiping [16, 42, 66, 131], like in the variant of PL-Gossip presented in the previous chapter. Since PL-Gossip can work with either of these approaches, both of them were implemented in the routing library. Moreover, their hybrid was implemented as well, which illustrates how one can use the library to explore the design space of hierarchical routing. Due to the scarceness of studies on the impact of hierarchy information propagation mechanisms on the performance of hierarchical routing, the performance differences between and the trade-offs in these three different approaches are studied further in this chapter.

4.2.2. Routing Protocol Scheme

The above sample of the design points illustrates that an implementation of hierarchical routing involves many intricate design decisions. Since studying all such decisions is virtually impossible, as mentioned before, this chapter focuses
on the two major ones that directly affect the state, stretch, and robustness of a hierarchical routing protocol. These two are the hierarchy type and the information propagation mechanisms.

To this end, hierarchical routing in the library has been implemented according to the scheme described below. Essentially, the scheme is based on the design of *PL-Gossip*, but the decisions at the two design points of interest are left open.

**Principal Operation**

Link quality estimation and packet forwarding are precisely as in *PL-Gossip* (see Section 3.5 and Section 3.3 respectively) except for the next-hop candidate lookup, which depends on either of the hierarchy types and which is described in detail in Section 4.2.3. Routing state maintenance, in turn, consists of generalized versions of the mechanisms introduced by *PL-Gossip*, so that they can support different hierarchy types and hierarchy information propagation methods — the two design points of interest.

More specifically, the mechanisms for maintaining the cluster hierarchy (i.e., node labels and routing tables) operate in rounds, which are local for each node. In every round, a node is allowed to issue (broadcast) one message that advertises the cluster the node is the head of and propagates any label updates for this cluster. The nodes receiving the message refresh their routing entries for the cluster, adopt any label updates if they are members of the cluster, and possibly rebroadcast the message. At the end of its round, each node analyzes its routing table to learn about any changes in the network that have occurred since the last round. If the changes require hierarchy modification, the node updates its label locally. Such a local label update is then propagated to the affected nodes in subsequent messages issued by the node. This simple mechanism is used both for synthesizing and maintaining node labels, which is described next.

**Hierarchy Construction**

Initially, each node is a top-level head of its level-0 singleton cluster. Hence, the label of a node consists only of the identifier of the node. Whenever a top-level head discovers in its routing table an entry for another cluster head at the same or a higher level, it must either spawn a new higher-level cluster itself or join the higher-level cluster of the other node. In the first case, it would extend its label with its own identifier, promoting itself to a higher-level cluster head. In the second case, it would extend its label with the identifier of the other node.
Joining an existing cluster is preferred, as it decreases the number of clusters at subsequent levels. However, depending on the hierarchy properties, such as the distance to the cluster head or the cluster adjacency, joining may not always be possible.

When no joining is possible, all cluster heads must not promote themselves to higher levels at the same time, as this would not guarantee the exponential drop in the number of clusters at subsequent levels. Hence, a head probabilistically defers its promotion by drawing a random promotion time slot, $s$, and then waiting for $s$ time slots. If within these $s$ time slots other nearby cluster heads promote themselves, the head may join its cluster to one of their clusters; otherwise, the head promotes itself. To ensure that a head deferring a promotion learns timely about newly spawned higher-level clusters, the time slot at level $i$ is longer than the propagation time of a cluster advertisement from a level-$i$ head, which is proportional to the possible diameter of the cluster.

A cluster head that extended its label, either by spawning its own or joining an existing higher-level cluster, embeds the label update in its subsequent message. In this way, the members of the cluster can also update their labels consistently to ensure recursiveness. Moreover, other heads deferring a promotion can learn about the new cluster.

Hierarchy Recovery

When a cluster head has died, it no longer issues messages advertising its cluster. As a result, other nodes do not refresh the routing entries for that cluster. If a node has not received a cluster advertisement refreshing an entry for a certain number of rounds, the entry is evicted from the routing table. If a level-$i$ cluster head discovers that the entry for its parent level-$i+1$ cluster has been evicted, it concludes that its cluster must not be a subcluster of the no longer existing level-$i+1$ cluster. Consequently, it cuts its label down to level $i$. Later, by virtue of the above hierarchy construction mechanisms, the disconnected cluster of this node will join some other higher-level cluster, thereby restoring the hierarchy.

Summary

Although the above scheme is largely based on the design of PL-Gossip, it leaves the two aforementioned design points open. As mentioned above, these design points have a major impact on the state, stretch, and robustness of hierarchical routing. They also differentiate the proposed hierarchical routing protocols. Therefore, when evaluating the implementation of this scheme with different solutions at these design points, not only PL-Gossip, but also solutions
adopted in other proposed hierarchical routing protocols will be evaluated, which corresponds to studying to some extent the design space offered by hierarchical routing.

### 4.2.3. Design Point: Hierarchy Type

The first design point left open in the above scheme is the type and the actual properties of the cluster hierarchy used for routing. As mentioned in Section 4.2.1, there are two major hierarchy types: an area hierarchy (AH) and a landmark hierarchy (LH). In short, an area hierarchy is conceptually created by grouping nodes into areas, grouping areas into superareas, and so on. A landmark hierarchy, in turn, is created by appointing some nodes as various-level landmarks and binding other nodes to the closest landmark at each level. Table 4.1 presents the hierarchy types adopted in selected protocols for hierarchical routing proposed to date.

<table>
<thead>
<tr>
<th>Hierarchy Type</th>
<th>Acronym</th>
<th>Some Protocol Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>area hierarchy</td>
<td>AH</td>
<td>Kleinrock &amp; Kamoun [69], Hagouel [52], Safari [22], Subramanian &amp; Katz [122], PL-Gossip</td>
</tr>
<tr>
<td>landmark hierarchy</td>
<td>LH</td>
<td>Tsuchiya [131], SCOUT [75], Bandyopadhyay &amp; Coyle [5], L+ [16], PL-Gossip</td>
</tr>
</tbody>
</table>

Table 4.1: Different hierarchy types in proposed protocols. The table presents the hierarchy types adopted in proposed representative hierarchical routing protocols. PL-Gossip can support both landmark and area hierarchies.

To a large extent, these two hierarchy types are similar. In particular, due to explicitly defining cluster head in an area hierarchy, they can both adopt the same definition of a routing address (i.e., the label): the label of a node is a concatenation of the identifiers of the heads of the clusters the node belongs to at each level, starting from level 0. Likewise, the general structure of a routing table and the fields of a routing entry are the same for both hierarchy types: a routing table consists of rows corresponding to hierarchy levels; each row contains entries for nearby clusters at the corresponding level; each entry for a cluster contains the level of the cluster, the unique identifier of the head, a list of next-hop neighbors on the shortest path from the node to that cluster, the number of hops on this path, and some maintenance fields, as explained in detail in Section 3.2.3. Finally, the principal idea behind routing in these two hierarchy types is also the same: to route a packet toward clusters at decreasing levels, which corresponds to the packet getting closer to its destination.
However, there are some subtle differences in the details of the two hierarchy types, notably in their rules for storing node routing entries and in the corresponding next-hop lookup algorithms. Since these differences affect routing state, routing stretch, and robustness, they are discussed in more detail below.

Differences in Routing Table Rules

To begin with, the rules the two hierarchy types introduce for deciding whether an entry for a cluster should be present in a routing table of a node differ.

In an area hierarchy, a cluster is determined by its members and can be identified arbitrarily. In other words, even though PL-Gossip dynamically selects one node in each cluster (area) as a head and uses the identifier of the head to identify the cluster, the definition of an area hierarchy does not necessitate this. Since the definition of an area hierarchy does not inherently involve any special nodes, the rules for storing a routing entry at a node are based solely on the label of the node. More specifically, in an area hierarchy, a node stores routing entries for those level-$i$ clusters the members of which have their label suffixes equal to the label suffix of the node starting from level $i+1$. Such clusters are simply level-$i$ siblings of the cluster of the node in the label tree; hence, this rule is simply a reformulation of the rule from Section 3.2.3.

One consequence of this rule is that, in an area hierarchy, the boundary for storing a routing entry for a level-$i$ cluster is the boundary of the parent level-$i+1$ cluster, as determined from node labels. This means, in particular, that even if the diameter, $D(i+1)$, of the parent level-$i+1$ cluster is smaller than the maximal possible diameter (e.g., $D(i+1) \ll 3^i - 1$ for the sample PL-Gossip area hierarchy from Section 3.2.1), routing entries for a level-$i$ cluster will not be stored by nodes more than $D(i+1)$ hops apart, even though the distance of such nodes from the cluster may still be much smaller than the maximal possible cluster diameter (i.e., $3^i - 1$ hops in the example). Another consequence is that a routing entry for a cluster in an area hierarchy does not need to point to the same cluster member at each node. Conversely, the routing entries for the same cluster at two different nodes may contain information about the shortest paths to two different members of the cluster. This may be exploited to improve the robustness of a routing protocol using area hierarchy.

In contrast, in a landmark hierarchy, the definition of a cluster is implicit as each cluster is directly related to some landmark: a landmark is the head of its virtual cluster and identifies the cluster; the cluster, in turn, consists of the nodes bound to the landmark. For this reason, the rules for storing a routing entry for a cluster at a node are directly related to the cluster head (the landmark)
and are based on the distance from the node to the head. More specifically, in a landmark hierarchy, a node stores routing entries for those level-i clusters the heads of which are up to $R(i) = \alpha^i \cdot \alpha_0$ hops away from the node (e.g., $R(i) = 2^i$), irrespective of the parent clusters of these clusters.

One immediate consequence of this rule is that, in a landmark hierarchy, a routing entry for a cluster always points to the head of the cluster at each node. Another consequence is that the boundary for storing a routing entry for a level-i cluster is simply the maximal number of hops, $R(i)$, from the head of the cluster. This means that even nodes that do not belong to the cluster or to its parent cluster but are within $R(i)$ hops from the cluster head will store the entry for the cluster, irrespective of their labels. It also means, however, that not all nodes within $R(i)$ hops from the head are allowed to be in the cluster of the head. More specifically, hierarchical routing requires that at least the head of each level-i cluster has routing entries for all its level-i−1 child clusters, which implies that, in a landmark hierarchy, the heads of child clusters have to be within $r(i) \leq R(i-1)$ hops from the head of their parent level-i cluster. All in all, in a landmark hierarchy, in contrast to an area hierarchy, many nodes that do not belong to a cluster or a parent cluster of the cluster may still store routing entries for the cluster.

**Differences in Next-Hop Lookup Rules**

These differences in the routing table rules affect the way in which subsequent routing hops are selected in the two hierarchy types.

An area hierarchy guarantees that any path between two nodes from the same cluster consists only of nodes from the cluster, as formalized by Lemma 3.1. This means that, for any level $i$, when a routed packet enters a level-i cluster of its destination node, first, it will never leave the cluster, and second, it will find an entry for a level-i−1 cluster of the destination. Entering lower-level clusters of the destination corresponds to the packet getting closer to the destination itself.

With such guarantees, selecting a next routing hop in an area hierarchy is performed based on the level of the common cluster of the current forwarding node and the destination node, as presented in Listing 3.3. When a packet arrives at a node that shares a level-i cluster with destination node, it is immediately redirected toward a lower-level cluster of the destination, as a routing entry for such a cluster is guaranteed to exist at the forwarding node. In this way, if the hierarchy is correct, the packet is guaranteed to reach the destination.

In contrast, a landmark hierarchy does not give any guarantees that a path between two nodes from the same cluster consists only of nodes from the cluster. Conversely, because of the rules for storing routing entries in a landmark
function lookupNextHopCandidates(
    hdr : RoutingPktHdr) : NextHop[];

var
    i : integer;
    rtEntry : RoutingEntry;
begin
    if hdr.dstAddr[0] = this.addr[0] then
        return [this.meAsNextHop]
    else begin
        i ← 1;
        while (i < hdr.dstAddr.len) do begin
            rtEntry ← lookupInRoutingTable(i, hdr.dstAddr[i]);
            if rtEntry ≠ null then return rtEntry.nextHopNeighbors;
            i ← i + 1;
        end;
        return [];
    end;
end;

Listing 4.1: The next-hop lookup routing function for a landmark hierarchy. It finds the cluster head toward which the packet can be forwarded. This is the head of the lowest-level cluster for which the forwarding node has a routing entry and to which the destination node belongs. A list of next-hop neighbors on the shortest paths to such a head is returned (as associated with the routing entry for the cluster of the head). The above function assumes a lack of failures; it gets more intricate when failures may occur.

hierarchy, it is often the case that the shortest path from a node in a cluster to the head of the cluster involves nodes that do not belong to the cluster. This means that, on its way to the destination, a routed packet that has entered a level-$i$ cluster of the destination may leave and re-enter the cluster a number of times.

This changes slightly the manner in which a next routing hop is selected in a landmark hierarchy, as presented here in Listing 4.1. More specifically, in a landmark hierarchy, the next-hop lookup function ignores the label of the forwarding node. Instead, it searches the routing table of the node for the lowest-level cluster of the destination. If such an entry is found at level $i$, all subsequent forwarding nodes are guaranteed to find the same entry at level $i$ or lower levels and, ultimately, if the hierarchy is correct, the packet is guaranteed to reach the destination.

As mentioned in Section 3.3, an area hierarchy does not introduce any inherent routing hot spots at cluster heads. As soon as a packet enters any node in a level-$i$ cluster of the destination, it is redirected toward a lower-level cluster, that is, it does not have to reach the head of the level-$i$ cluster. Likewise, even though it forwards packets toward cluster heads rather than arbitrary cluster members, the algorithm for a landmark hierarchy does not introduce inherent hot spots either. As soon as a packet routed toward a level-$i$ cluster head of the destination enters
4.2. IMPLEMENTING HIERARCHICAL ROUTING

a node within $R(i-1)$ hops from the level-$i-1$ cluster head of the destination, it is redirected toward the level-$i-1$ head. Chen and Morris present an analysis of routing load balancing for hierarchical routing in a landmark hierarchy [16].

Summary

The above differences in the rules for storing node routing entries and in the corresponding next-hop lookup algorithms result in the differences in the performance of hierarchical routing in the two hierarchy types. Tsuchiya demonstrated analytically that the routing state in a landmark hierarchy is typically larger than the state in an area hierarchy [131]. This is because a node in a landmark hierarchy typically stores more entries for clusters it does not belong to than a corresponding node in an area hierarchy. Conversely, a landmark hierarchy typically offers a smaller stretch than an area hierarchy. This is because, in a landmark hierarchy, a node stores more routing entries and a path between two nodes in a cluster does not have to involve only nodes from the cluster, and, thus, packets can be forwarded over shorter paths than in an area hierarchy [131]. One can further exploit the state-stretch trade-off in hierarchical routing by not only varying the hierarchy type, but also the properties of the hierarchy, for example, the cluster scaling function $R(i) = \alpha^i \cdot \alpha_0$ in a landmark hierarchy.

To study the hierarchy type design point, support for both types has been included in the hierarchical routing library. For the experiments with an area hierarchy (AH), the sample hierarchy introduced for PL-Gossip in the previous chapter was chosen. For the experiments with a landmark hierarchy (LH), in turn, support for different cluster scaling functions, $R(i)$, was implemented. To make the two hierarchy types more similar, in an area hierarchy, a path to a cluster in a routing entry was made to always describe a path to the head of the cluster rather than to an arbitrary node within the cluster, which is consistent with the definition of a routing entry in a landmark hierarchy. All in all, the library provides means for studying the performance of hierarchical routing in hierarchies of the two types and of various properties.

4.2.4. Design Point: Propagating Hierarchy Information

The second major design point left open in the routing protocol scheme described in Section 4.2.2 are the mechanisms for propagating up-to-date information about clusters in the network. According to that scheme, to diffuse label updates (extensions and cuts) and to advertise its cluster to other nodes, in every round each node issues (broadcasts) a message. The pattern according to which such messages are issued and the manner they propagate hierarchy information
Table 4.2: Different update propagation methods in proposed protocols. The table presents different update propagation mechanisms adopted in proposed representative hierarchical routing protocols. While the variant of PL-Gossip presented in the previous chapter uses local gossiping, this mechanism can be replaced with the any of the other two, as is illustrated in the hierarchical routing library. In other words, update propagation mechanisms are interchangeable.

determine the latency and the traffic required to bootstrap a hierarchy and to recover it after failures. Table 4.2 presents the major hierarchy information propagation methods. The first two of these methods have been adopted in hierarchical routing protocols proposed to date; the third one, invented during the course of this research, illustrates how the hierarchical routing library allows for exploring the design space.

Since these mechanisms determine the latency and the traffic required to bootstrap a hierarchy and to recover it after failures, they directly affect the robustness of a hierarchical routing protocol. For this reason, they constitute an important design point, which, however, has not been thoroughly studied to date. To fill this gap in, the mechanisms are described in more detail below.

Local Gossiping

The first method, local gossiping, is used, among others, by the variant of PL-Gossip presented in the previous chapter. In essence, at a random moment in every round, each node broadcasts its routing state in a heartbeat message. Such a message is received by the neighbors of the node and is not forwarded by them. The broadcast state contains the label of the node, the routing table, and the update vector; the latter allows for propagating label updates as described in Section 3.4.3. The neighbors receiving the message refresh their routing tables using a hierarchical distance-vector algorithm, in which the rules determining the propagation scope for a cluster advertisement are as described in Section 4.2.3 for different hierarchy types. The neighbors can also adopt any fresh label updates performed by the heads of the clusters they share with the heartbeat issuer, which is performed using the label combine operation, explained in Section 3.4.3.

In local gossiping, hierarchy information thus propagates implicitly.
Heartbeat messages are local, that is, a heartbeat message is received only by the neighbors of its sender and these neighbors do not forward it. Instead, they merge the received state with their own local state, and rebroadcast such a fresher state in their own heartbeats in subsequent rounds. Therefore, hierarchy information propagates over multiple hops due to nodes periodically exchanging and merging their local state rather than due to explicitly forwarding packets.

If the state of a node does not fit in a frame, it has to be fragmented. Local gossiping has been implemented with two forms of fragmentation: first, the MAC layer fragments a heartbeat message into multiple frames that are sent after a single preamble \((Gossip[sf])\), and second, the protocol fragments its state into multiple one-frame heartbeat messages, which are broadcast independently, each with its own preamble \((Gossip[m])\).

**Multi-Level Scoped Flooding**

In the second method, multi-level scoped flooding, to advertise its cluster, each head periodically issues (broadcasts) a beacon message. A beacon message contains the label of the cluster head, a sequence number, and a hop count, hence it is much smaller than a heartbeat message in local gossiping (it does not contain the routing table of the issuing node). A node receiving the beacon refreshes the routing entry for the cluster of the head, adopts any label updates performed by the head (if it belongs to the cluster), and rebroadcasts the beacon if it is within the advertisement scope of the cluster, as described in Section 4.2.3.

Therefore, in contrast to a heartbeat message, which is local and propagates information about multiple clusters, a beacon message travels over multiple hops and is dedicated to a particular cluster. A beacon message issued by the head of a cluster is flooded in the network by all nodes within the advertisement scope of the cluster, that is, all nodes in the advertisement scope that receive the beacon, rebroadcast it further. For example, in a landmark hierarchy, all nodes within \(R(i) - 1\) hops from a level-\(i\) cluster head rebroadcast beacons issued by this head. In other words, in multi-level scoped flooding, hierarchy information propagates explicitly, by forwarding beacon messages over multiple hops.

Since each beacon is dedicated to a single cluster and is typically flooded over multiple hops, multi-level scoped flooding can be costly in terms of the number of transmitted messages. To alleviate this, often the inter-beacon interval of the head of a cluster is proportional to the advertisement scope of the cluster, that is, it grows exponentially with the level of the cluster. For example, in a landmark hierarchy, a level-0 head issues a \(R(0)\)-hop beacon every \(R(0)\) rounds, a level-1 head — a \(R(1)\)-hop beacon every \(R(1)\) rounds, and so on. This amortizes the
high costs of forwarding higher-level beacons over many rounds, but increases the latency of bootstrapping and recovering the hierarchy. To mitigate this increase, a cluster head is also allowed to issue a beacon immediately after it has changed its label locally. Both variants have been implemented: with \((\text{Flood}[e])\) and without \((\text{Flood}[c])\) the exponentially increasing inter-beacon period.

Hybrid Method

As essentially a flooded beacon can be immediately rebroadcast by a receiving node, multi-level scoped flooding can propagate information quickly. However, since each beacon is dedicated only to a single cluster and has to be flooded over a number of hops, when performed periodically, multi-level scoped flooding results in myriads of inefficient short transmissions. While increasing the beacon issuing interval exponentially with the level alleviates this problem, it introduces its own set of problems. With the exponential beacon issuing interval, if a node misses a beacon for a high-level cluster, it may not be able to route to the members of the cluster for a long time.

In contrast, as the routing entries in a single heartbeat message advertise many clusters, local gossiping generates lower traffic. In addition, even if in some round a node misses a heartbeat with a cluster advertisement, it will likely receive the advertisement in subsequent rounds. However, since local gossiping propagates information by merging the state of neighboring nodes only once per round, it may take up to \(R\) rounds to propagate an advertisement over \(R\) hops. Therefore, the latency of update propagation in local gossiping is inferior to the latency in multi-level scoped flooding.

The hybrid method \((\text{Hybrid})\) proposed here aims to combine the advantages of these two methods. In \(\text{Hybrid}\), when a cluster hierarchy is stable, nodes run local gossiping, thereby generating lower traffic. However, when a cluster head changes its label, for instance, as a result of some failure, it issues a beacon message to rapidly propagate the change among the members of its cluster or to advertise a new cluster. In this way, hierarchy bootstrapping and recovery after failures can be faster and, on average, the traffic can be low as well.

4.2.5. Implementation Summary

To sum up, due to the peculiarities of sensornets, evaluating a sensornet point-to-point routing protocol requires experiments with an actual implementation. Instead of implementing just the proposed \(\text{PL-Gossip}\) protocol, an entire hierarchical routing library has been developed. The library defines a common scheme for existing proposals for hierarchical routing and at
the same time identifies various design points that differentiate the proposals. Therefore, by varying the solutions at these design points, one can experiment not only with an implementation of PL-Gossip, but also, to some extent, with implementations of other existing proposals for hierarchical routing. To the best of my knowledge, the library is the first reported implementation of hierarchical routing for sensornets.

Two crucial design points that heavily influence the routing state, stretch, and robustness of a hierarchical routing protocol are the type of the employed hierarchy and the methods for propagating hierarchy information. To study the first design point, two common hierarchy types have been implemented in the library: an area hierarchy (AH) and a landmark hierarchy (LH). For each of the types, one can also customize the actual properties of the hierarchy. To study the second design point, three methods for propagating hierarchy information have been implemented; the first two, multi-level scoped flooding (Flood) and local gossiping (Gossip), are common existing methods; the third one, a hybrid of the two (Hybrid), is a new method introduced here. Again, the library contains multiple variants of each of the methods. The next section studies how the above decisions at the two crucial design points affect the performance of hierarchical routing.

4.3. LOW-LEVEL IMPLEMENTATION-BASED SIMULATION

The first platform on which the implementation of the hierarchical routing library was evaluated was a low-level simulator, TOSSIM 2.0. TOSSIM runs virtually the same TinyOS code as actual sensor nodes and simulates node hardware components at a low level. It also incorporates realistic signal propagation models of low-power wireless communication, which have been derived from real-world deployments. Therefore, low-level implementation-based simulation in TOSSIM aimed at evaluating the scalability of hierarchical routing in realistic networks.

To conduct low-level simulation experiments, a number of representative topologies were generated. Like in Section 4.1 for illustrative purposes, just the two extremes of the employed node placement strategies, \( p \), are presented here: very regular (grid) and completely irregular (random). To study the protocol scaling properties, the number of nodes, \( N \), was varied exponentially from 64 to 1024, resulting in diameters of up to 15–18 hops in 1024-node networks. To also investigate the impact of node density, \( \rho \), the size of the deployment area was varied as well, yielding different node densities from \( \sim 11 \) (sparse) to \( \sim 48 \) (dense)
high-quality neighbors per node on average. The resulting network configurations cover a number of representative deployment scenarios.

Using the tools available with TOSSIM and real-world signal strength traces, for each configuration a realistic connectivity environment was generated. In those environments, there were many asymmetric links and nearby nodes often could not communicate — phenomena that are common in the real world (cf. Chapter 2). All in all, the TOSSIM results should predict the real-world behavior of hierarchical routing well.

4.3.1. Routing State

The principal motivation behind hierarchical routing is to offer small routing state; consequently, again the first property studied in the experiments is the size of the routing table maintained by a node.

Assuming a constant size of a unique node identifier, the size of a single routing entry in hierarchical routing can be considered constant. In particular, in the hierarchical routing library, a routing entry at a node needs at least 8 bytes for the fields listed in Section 4.2.3 (depending on the number of next-hop candidates) plus 4 bytes of overhead for a hash table. A routing entry transmitted in a heartbeat message, in turn, is compressed to 4 bytes. Therefore, again as the metric of the routing state the number of routing table entries is used.

Figure 4.7 depicts the number of routing entries obtained for area hierarchies in different network configurations. Each data point corresponds to the average (including the standard deviation) or the 99-th percentile over all nodes and 10 protocol runs, which resulted in 10 different hierarchies, each with the maximal level of 5, as this was enough for the top-level cluster to cover all nodes.

In accordance with the basic idea behind hierarchical routing, the routing state obtained with the library is small. More specifically, it appears to scale polylogarithmically with the network size, \( N \), (the left plot) with small factors in the polynomial of the logarithm. For example, in the sparse 1024-node network in the figure, a node requires on average less than 30 routing entries. Likewise, in 99% of the cases the routing state is also small. For instance, in the sparse 1024-node network in the figure, the 99-th-percentile state does not exceed 40 entries, that is, it is only 33.3% larger than the average state.

The analytical results cited in the previous chapters prove that such a polylogarithmic state is possible, provided that the cluster hierarchies are optimal [66]. In PL-Gossip and the hierarchical routing library, however, the hierarchies are built using only probabilistic heuristics, and thus, they are likely not optimal. Nevertheless, the routing state they induce still appears polylogarithmic.
4.3. Low-Level Implementation-Based Simulation

Figure 4.7: The routing table size (TOSSIM). The figure depicts the average (including the standard deviation) and the 99-th-percentile number of routing entries per node for exponentially growing node populations (left) and varying node densities (right) in networks with realistic low-power wireless communication. For every network size, the aggregates have been computed over 10 independent simulation runs, each resulting in a different area hierarchy satisfying the sample properties from the previous chapter. The figure shows that, like in idealized unit-disk networks from Figure 4.2, the routing state maintained by nodes in realistic networks is small. Overall, the figure indicates that not only in theory, but also in practice can hierarchical routing offer small routing state.

This suggests that the mechanisms introduced in PL-Gossip and the library to enable hierarchical routing in sensornets work well in realistic networks. Even though they have to deal with the resource constraints of sensor nodes and the peculiarities of low-power wireless communication, they build and maintain hierarchies that result in small routing state.

Counter-intuitively the routing state in realistic networks from Figure 4.7 seems smaller than the state in unit-disk networks from Figure 4.2. There are two reasons for such a difference.

First, in TOSSIM experiments, the top hierarchy level was fixed to 5. That meant that even though there might not have been a single top-level cluster at level 5, no additional clusters were created at higher levels, which in turn reduced the number of routing entries the nodes had to maintain. Bounding the maximal hierarchy level bounds the length of the routing address and is possible when the maximal network diameter is known. The only modification required to guarantee successful routing in such a hierarchy is requiring the top-level clusters to advertise themselves among all nodes. In other words, a bounded hierarchy assumes that all top-level clusters belong to the same (nonexistent) higher-level cluster. Overall, such a modification is trivial and requires just a few “if…then…” statements in the hierarchy maintenance code.
Second, in the hierarchical routing library, the following cross-layer optimization was applied. Since the link table of a node, that is, the table with the estimates of the link quality for the neighbors, contains all nodes within one hop from the node, the link table can be used for routing. More specifically, a node does not need to maintain redundant routing entries for level-0 clusters, and instead, it can use its link table when routing to such clusters. Although in both area and landmark hierarchies there may be sibling level-0 clusters that are up to two hops away from the node, the node can always route to such clusters via its level-1 cluster head. Alternatively, a more general recovery method, implemented in the S4 routing protocol [94], can be used: the node broadcasts the packet destined to a sibling level-0 cluster that is two hops away; any neighbor that has an entry for that target cluster acknowledges the reception and forwards the packet; a node can store an entry with that neighbor as the next hop temporarily for the current routing flow. Such a cross-layer optimization eliminates redundancy between the link table and routing table and is common in memory-constrained sensor nodes [27, 47, 105]. If there were no such an optimization, the actual number of routing entries would be at most the sum of the number of entries from Figure 4.7 and the number of neighbor entries from the same figure; it might be lower because not all neighbors of a node belong to the same level-1 cluster as the node.

The two above engineering decisions yield the reduction in the node routing table sizes as compared to high-level simulation with unit-disk connectivity. Especially in denser networks, they make the routing tables even smaller, as can be observed in the right plot of Figure 4.7. This is because in denser networks each node has more nodes within one hop, and thus, does not need to maintain routing entries for such nodes. If the same decisions were used in simulation with realistic and unit-disk wireless communication models, however, the routing tables obtained in more regular unit-disk topologies would be smaller, as was shown in an earlier paper that contributed to the results of this chapter [55]. The example of these two engineering decisions thus illustrates that there is still room for improving the mechanisms introduced by PL-Gossip.

All in all, the results demonstrate that hierarchical routing, in general, and the mechanisms introduced by PL-Gossip and the library, in particular, can indeed offer small routing state in realistic sensornet deployments, like in idealized unit-disk networks. Like in unit-disk networks, in realistic networks, the number of routing entries maintained by a node in hierarchical routing is, on average, close to the logarithm of the total node population size, which ensures excellent scalability. Moreover, the 99-th-percentile routing state is close to the average, in particular, it is much lower than twice the average. Therefore, when
provisioning node memory pools for routing entries, one can expect that a node on average utilizes more than 50% of its pool. Overall the above results for realistic networks confirm that, from the routing state perspective, hierarchical routing also in practice is indeed an excellent point-to-point routing technique for resource-constrained sensor nodes.

4.3.2. Routing Stretch

The state aggregation in hierarchical routing reduces the size of node routing tables. However, it also lengthens the resulting routing paths. Therefore, the second property studied with low-level simulation is the extent to which the routing paths increase, that is, routing stretch.

Like in the high-level simulation experiments, the routing stretch is measured with the standard metric: hop stretch. The hop stretch of a routing path between two nodes is the ratio of the number of hops on this routing path to the number of hops on the shortest possible path between the two nodes in the internode connectivity graph. However, unlike in unit-disk networks, in which a link in the internode connectivity graph either offers perfect packet reception or does not exist at all, in real-world networks the behavior of a virtual wireless link is more intricate: real-world wireless links exhibit packet loss that may vary spatially and temporarily. For this reason, the original definition of the hop stretch metric needs to be modified for low-power wireless networks. More specifically, in such networks, a hop is typically defined over a wireless link with the bidirectional packet reception rate, as measured by the employed link quality estimator, exceeding some threshold (55% in the experiments). In other words, such a definition does not consider marginally bad links for stretch measurements.

Figure 4.8 depicts the hop stretch in the hierarchies from Figure 4.7. Like in idealized unit-disk networks, in realistic networks hierarchical routing offers small hop stretch that scales gracefully with the network size (the left plot). For example, in the sparse networks in the figure, the average hop stretch never exceeds 1.5. Likewise, the hop stretch does not grow significantly with the increase in the node density (the right plot). In particular, in all networks of varying densities presented in the figure, the average hop stretch remains close to 1.5. This means that despite maintaining only a small state involving a polylogarithmic number of routing entries, hierarchical routing in area hierarchies can in practice offer routing paths that are within 50% of the optimal ones on average. It will shown shortly that the results are even better in landmark hierarchies.

Moreover, although the theoretical upper bound on the hop stretch between
The routing stretch (TOSSIM). The figure depicts the average (including the standard deviation) and the 99-th-percentile hop stretch in the area hierarchies from Figure 4.7. The figure shows that, like in idealized unit-disk networks from Figure 4.3, the hop stretch of the routing paths in the hierarchies built in realistic networks is small. This can be explained by the fact that, due to embedding in physical space, the diameters of sensornets grow fast. In networks with fast-growing diameters, the stretch of hierarchical routing can remain small. This figure combined with Figure 4.7 thus indicates that despite maintaining small routing state, hierarchical routing can in practice offer small routing stretch.

**Figure 4.8:** The routing stretch (TOSSIM). The figure depicts the average (including the standard deviation) and the 99-th-percentile hop stretch in the area hierarchies from Figure 4.7. The figure shows that, like in idealized unit-disk networks from Figure 4.3, the hop stretch of the routing paths in the hierarchies built in realistic networks is small. This can be explained by the fact that, due to embedding in physical space, the diameters of sensornets grow fast. In networks with fast-growing diameters, the stretch of hierarchical routing can remain small. This figure combined with Figure 4.7 thus indicates that despite maintaining small routing state, hierarchical routing can in practice offer small routing stretch.

Two nodes in hierarchical routing is high [66, 70]. Figure 4.8 shows that more than 99% of routing paths in the conducted experiments do not exceed a hop stretch of 3.5. Moreover, the worst paths obtained during the experiments had a hop stretch of 6. This suggests that although in theory one can presumably construct a hierarchy with a high-hop-stretch path, in practice, the hierarchies synthesized with the presented mechanisms offer small-hop-stretch paths with very high probability. Again the explanation for the small hop stretch is that, due to embedding in physical space, the internode connectivity graphs of sensornets are “geometric,” that is, their diameters grow fast with the node population. In such networks, the state aggregation due to clustering does not impair routing stretch significantly. As a result, despite maintaining few routing entries, hierarchical routing can offer small hop stretch in practical sensornet deployments.

However, since the definition of hop stretch adopted for low-power wireless networks does allow links with some packet loss, it may describe the real cost of a routing path inaccurately. More specifically, since the hierarchy is built upon links with non-zero packet loss, the total number of transmissions necessary to deliver a packet over a routing path may be higher than the number of hops on this path. In particular, a routing path may consist of few hops but may involve poor, lossy links, which overall results in many (re)transmissions when routing a packet over this path. To quantify the discrepancy between the hop stretch and the
actual transmissions, another standard metric for low-power wireless networks is used: transmission stretch. The transmission stretch of a routing path between two nodes is the ratio of the number of transmissions to deliver a packet using the path to the number of hops on the path. When measuring transmission stretch in the experiments, the packet loss due to collisions and congestion was minimized by routing only one packet at any given moment.

The transmission stretch results (not plotted) indicate that in the hierarchical routing library the discrepancy between the hop stretch of a path and the actual transmission cost of the path is small; the average transmission stretch is approximately 1.02 and the 99-th percentile is not greater than 2. The low transmission stretch is mainly a consequence of the mechanisms used for selecting next-hop neighbor candidates for routing entries and for choosing the actual candidates when forwarding packets. The framework uses neighbor tables that are large enough to allow each node to select high-quality links as routing hops. Moreover, link quality estimates require several samples to converge. In effect, due to the bimodality of wireless links [118, 119, 138], even though a hop was defined over a link with packet reception of at least 55%, most of the hops in fact exhibited nearly 100% packet reception, which effectively minimized the transmission stretch.

Taking everything into account, like in idealized unit-disk networks, the results in realistic networks suggest that hierarchical routing, in general, and the mechanisms introduced by PL-Gossip and the library, in particular, can offer small routing stretch. In the experiments, the average hop stretch was approximately 1.5 for an area hierarchy and the 99 percentile did not exceed 3.5. These results can be supported analytically by the fact that sensornets are typically “geometric,” which means that despite routing state aggregation, routing stretch does not need to grow. Moreover, the actual number of transmissions necessary to deliver a packet over a routing path produced by the hierarchical routing library is also very small. This, in turn, is the consequence of the mechanisms that choose high-quality links when building and selecting routing paths. Overall, the above results confirm that, from the routing stretch perspective, hierarchical routing is also attractive as a point-to-point routing technique for practical sensornet deployments.

### 4.3.3. Comparison of Area and Landmark Hierarchies

The state-stretch trade-off in hierarchical routing can further be explored. While the results were obtained on an area hierarchy (AH) from the previous chapter (i.e., one satisfying the four sample properties), the hierarchical routing library
also supports other hierarchies, in particular, a landmark hierarchy \((LH)\), as described in Section 4.2.3. Switching between different hierarchy types and further customizing the properties of a particular hierarchy can influence the amount of routing state the nodes maintain and the routing stretch they achieve with such a state.

Figure 4.9 compares the routing state in the sample area hierarchy \((AH)\) and three landmark hierarchies \((LH(\alpha))\) with different cluster scaling functions, \(R(i) = \alpha^i\). The settings for the experiments conducted to obtain the figure were the same as the settings for the above experiments for an area hierarchy.

The figure suggests that a landmark hierarchy typically requires a larger routing state than an area hierarchy. This is because of the differences in the rules for storing routing entries in the two hierarchy types, as explained in Section 4.2.3. In an area hierarchy, a routing entry for a cluster is stored only by the members of the cluster and the members of the sibling clusters in the supercluster. In contrast, in a landmark hierarchy, all nodes within \(R(i)\) hops from a level-\(i\) cluster head, irrespective of whether they belong to the cluster of the head, store routing entries for the cluster of the head. This means that, in contrast to an area hierarchy, many nodes in a landmark hierarchy that do not belong to a cluster or the supercluster of the cluster still store routing entries for the cluster. In effect, the routing state in a landmark hierarchy is typically larger than the routing state in a similar area hierarchy. This is consistent with earlier analytical results [131].

Another phenomenon visible in the figure (in the right plots) and associated with the rules for storing routing entries is that whereas in an area hierarchy the routing state shrinks with the node density, in a landmark hierarchy it grows. This phenomenon is best explained by means of an example.

Assume a fully-connected network with an \(H+1\)-level hierarchy, either a landmark hierarchy or an area hierarchy. In both the area and the landmark hierarchy, at the top level, \(H\), there is a single cluster for which each node maintains an entry; thus the base number of routing entries in each of the hierarchies is: \(AH:1\) vs. \(LH:1\). Suppose that at level \(H-1\) there are \(n_{H-1}\) clusters. In both hierarchies, each node maintains an entry for each of those clusters; thus the number of routing entries per node in each of the hierarchies grows to: \(AH:1 + n_{H-1}\) vs. \(LH:1 + n_{H-1}\). Suppose that at level \(H-2\) there are \(n_{H-2}\) clusters, such that each supercluster has \(n_{H-2}/n_{H-1}\) of those clusters as subclusters. In the landmark hierarchy, the advertisement radius of a cluster head is \(R(H-2) = \alpha^{H-2} \geq 1\); therefore, since the network is fully connected, each node maintains an entry for each of the level-\(H-2\) clusters, that is, \(n_{H-2}\) entries at level \(H-2\) in total. In contrast, in the area hierarchy, each node from
4.3. Low-Level Implementation-Based Simulation

Figure 4.9: The routing state in area and landmark hierarchies (TOSSIM). The figure depicts the average (top) and the 99-th-percentile (bottom) number of routing entries per node for exponentially growing node populations (left) and varying node densities (right) in networks with realistic low-power wireless communication. The settings for the experiments contributing to the figure were the same as the settings for the experiments contributing to Figure 4.7 and 4.8. The memory pool for routing entries had 250 slots, hence the cut at 250. AH denotes an area hierarchy satisfying the four sample properties from the previous chapter; \( LH[\alpha] \) denotes a landmark hierarchy with cluster scaling function \( R(i) = \alpha^i \). The figure confirms that a landmark hierarchy typically requires more routing entries than an area hierarchy, which was explained in detail in Section 4.2.3. In general, by customizing the hierarchy, one can obtain different routing state for hierarchical routing.

\[ \rho: \text{sparse}; \; p: \text{random} \]

\[ N: 1024; \; p: \text{random} \]

\[ \text{area hierarchy} \]

\[ \text{landmark hierarchy} \]

\[ \text{99-th perc. num. routing entries} \]

\[ \text{average neighborhood size} \]

\[ \text{average num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{average neighborhood size} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]

\[ \text{avg. num. routing entries} \]

\[ \text{avg. num. routing entries} \]

\[ \text{network size} \]
compared to the landmark hierarchy is a consequence of the rules for storing routing entries in those two hierarchy types, more specifically, of a so-called boundary effect [94]. A boundary effect reflects a situation in which nodes $A$ and $B$ are neighbors, but $A$ does not have an entry for $B$ in its routing table (and vice versa) because $A$ and $B$ are in different higher-level clusters. Such a situation is common in an area hierarchy, but does not occur in a landmark hierarchy. Consequently, in a landmark hierarchy, the number of routing entries grows with an increase in the network density, that is, in the number of neighbors of a node. In contrast, in an area hierarchy, not only can the boundary effect prevent such a growth, but can also allow for creating smaller hierarchies, which require even fewer routing entries in denser networks.

Finally, Figure 4.9 also shows that by customizing the actual properties of a hierarchy one can influence the size of the node routing state. In particular, by changing the base, $\alpha$, of the cluster scaling function, $R(i) = \alpha^i \cdot \alpha_0$, in a landmark hierarchy, one can control the amount of routing entries nodes have to maintain. Like the ones in the previous sections, these results are also consistent with earlier analytical results [66, 131]. A major contribution of those analytical results is identifying the optimal settings for each of the hierarchy types, that is, the minimal possible routing state in each of the hierarchy types. In this view, the analytical results complement the experimental ones presented here.

Overall, the above experimental results illustrate that hierarchical routing in either of the hierarchy types can offer small routing state. While an area hierarchy yields a smaller state than a landmark hierarchy, a landmark hierarchy with the most common cluster scaling function, $R(i) = 2^i$, results in just a slightly larger state. In particular, in the sparse 1024-node networks in Figure 4.9, a node in an area hierarchy (AH) requires approximately 29 routing entries on average, whereas a node in a landmark hierarchy with the common cluster scaling function (LH[2]) requires about 45 entries on average. Moreover, in both hierarchy types in 99% of the cases, the routing state of a node does not exceed twice the average state, which is a useful property when provisioning memory pools for node routing entries. Finally, a slightly larger routing state in a landmark hierarchy may actually not be a disadvantage in some applications as it reduces the routing stretch, which can be observed in Figure 4.10.

Essentially, combined with Figure 4.9, Figure 4.10 illustrates two major points. First, it shows the state-stretch trade-off within hierarchical routing itself: the larger the routing state, the smaller the routing stretch. Because of maintaining more routing entries, nodes can forward packets along paths that closer to the optimal ones. This in effect reduces routing stretch. Second, due to maintaining more routing entries, a landmark hierarchy offers even smaller stretch than an
Figure 4.10: The routing stretch in area and landmark hierarchies (TOSSIM). The figure depicts the average (top) and the 99-th-percentile (bottom) hop stretch for exponentially growing node populations (left) and varying node densities (right) in networks with realistic low-power wireless communication from Figure 4.9. AH denotes an area hierarchy satisfying the four sample properties from the previous chapter; LH[α] denotes a landmark hierarchy with cluster scaling function $R(i) = \alpha^i$. Like in Figure 4.9, the memory pool for routing entries had 250 slots, hence the cut in networks where the routing table exceeded 250 entries. Combined with Figure 4.9, this figure illustrates the state-stretch trade-off within hierarchical routing itself: the larger the state, the smaller the stretch, and vice versa. One can explore this trade-off by customizing the cluster hierarchy used for routing.

Area hierarchy. In particular, for LH[2] the average hop stretch is around 1.25 (vs. 1.5 for AH) and the 99-th percentile does not exceed 2 (vs. 3.5 for AH). This further reinforces the initial argument for using hierarchical routing in sensornets: in practical sensornet deployments hierarchical routing can simultaneously ensure small routing state and small routing stretch.

All in all, the above experiments comparing area and landmark hierarchies illustrate that by switching between different hierarchy types and customizing the properties of a particular hierarchy, one can explore the trade-off between routing state and routing stretch within hierarchical routing. Both landmark and area
hierarchies can be used to implement hierarchical routing in practical sensornet deployments. A landmark hierarchy typically requires more routing entries than an area hierarchy, but results in a smaller routing stretch. Nevertheless, irrespective of the hierarchy type, the state and stretch of hierarchical routing are small. Consequently, hierarchical routing is indeed an appealing technique when small routing state and small routing stretch are required simultaneously.

4.3.4. Robustness and Self-Management

Having demonstrated that hierarchical routing can simultaneously offer small routing state and small routing stretch, let us proceed to study dynamic aspects of this routing technique, notably the costs of bootstrapping and maintaining a cluster hierarchy. To this end, let us make use of the fact that a protocol in the hierarchical routing library operates in rounds. The duration of a round will thus be fixed and the maintenance costs will be studied in terms of rounds. Such an approach is accurate when a round lasts a few orders of magnitude longer than a message transmission. This is typically the case as in low-data-rate sensornet applications the rounds are measured in the order of seconds or even minutes (5 minutes in the experiments) while a message transmission takes in the order of milliseconds.

Hierarchy Construction

Figure 4.11 presents the number of rounds that different hierarchy information propagation methods require to bootstrap the hierarchies from Figure 4.9 and 4.10 excluding the time necessary for the link quality estimates to converge. Like in high-level simulation, in those experiments all nodes were started simultaneously and had to construct the hierarchy. The hierarchy was considered as being bootstrapped when 99% of the nodes had their routing addresses assigned and could successfully route to each other.

The hierarchy information propagation methods that offer the fastest bootstrap are multi-level scoped flooding with beacon messages issued in every round irrespective of the cluster head level \((\text{Flood}[c])\) and the novel hybrid approach \((\text{Hybrid})\). In these methods, the number of bootstrap rounds after the link quality estimates have converged is directly proportional to the maximal hierarchy level, 5. Essentially, in every round, a single hierarchy level is constructed (the right plot), which corresponds to the lower bound in the library. Consequently, these two methods are optimal with respect to the bootstrap latency.
Figure 4.11: The hierarchy bootstrap latency (TOSSIM). The figure depicts the average (over 10 independent runs) number of rounds necessary to fully bootstrap a cluster hierarchy using different hierarchy information propagation methods. This number does not include the few rounds necessary for the link quality estimates to converge. Flood[c] and Hybrid are the fastest methods because they can essentially build 1 level per round. In contrast, Gossip is the slowest one because it requires a number of rounds proportional to the network diameter (15 hops in the figure). Finally, although Flood[e] should also be fast in theory, due to packet loss it is slow in practice. In short, when a beacon message is lost, a node will not be able to route toward the head that issued the beacon for a long time, which boosts the bootstrap latency of Flood[e].

In theory, multi-level scoped flooding with the exponential beacon issuing pattern (Flood[e]) should perform like these two methods. In practice, however, it does not due to packet loss. If a node misses a beacon message from a level-i cluster head, it and potentially some of its neighbors will not record a routing entry for the head for \( R(i) \) rounds, which boosts the bootstrap latency. In the right plot of Figure 4.11, this phenomenon can be observed at level 4 \( R(4) = 16 \), at which some missed beacon from a level-4 cluster head delayed hierarchy construction at level 5 for 16 rounds, hence the peak in the hierarchy bootstrap time for a 1024-node network in the left plot. This can also happen in the two optimal methods. However, in Flood[c], the unlucky node will likely receive a beacon in the next round, whereas in Hybrid, the unlucky node will recover through heartbeat messages.

Finally, local gossiping (Gossip) is the slowest hierarchy construction method. Since in this method information is propagated through periodic state merging once per round, in the worst case, it may take \( R \) rounds to advertise a cluster over \( R \) hops. This also requires extending the slot duration in the cluster head promotion heuristics. As a result, the bootstrap latency depends mostly on the network diameter (15 hops in the figure).

With the traffic generated for bootstrapping, the relationship between the
hierarchy information propagation methods is opposite, as depicted in the left plot of Figure 4.12. In local gossiping, a node sends one heartbeat message per round, if a message can consist of a few frames (Gossip[s]), or a number of heartbeat messages that appears polylogarithmic, if the routing state is manually fragmented into one-frame heartbeat messages (Gossip[m]). Likewise, other methods generate apparently polylogarithmic traffic. The differences in the polynomials of the logarithms, however, can be substantial. For example, the difference between Flood[c] and Gossip[m] in the figure is a factor of 10.

Figure 4.12: The bootstrap traffic (TOSSIM). The figure depicts the average (over 10 independent runs) number of messages per round (left) and bytes per message (right) necessary to fully bootstrap a cluster hierarchy using different hierarchy information propagation methods. The relationship between the methods is the opposite of that for the bootstrap latency, as depicted in Figure 4.11. Gossip generates the lowest traffic that also consists of the longest messages. In contrast, Flood generates the heaviest traffic that involves myriads of inefficient short messages. Hybrid lays in between, in the bootstrap phase resembling Flood more, because in this phase beacon messages dominate over heartbeat messages.

For efficiency, message payloads should be maximized: a protocol should preferably send fewer but longer messages. This is crucial in sensornets because a message transmission or reception typically involves a large energy overhead due to synchronizing the transmitter and receivers to have their radios on during a transmission. The right plot in Figure 4.12 presents the efficiency of different hierarchy information propagation methods. Beacon messages are small (10 bytes in the library), and thus, multi-level scoped flooding (Flood) is potentially inefficient when bootstrapping the hierarchy. In contrast, heartbeat messages in local gossiping (Gossip) propagate information more efficiently, as each heartbeat contains the whole routing state of a node; consequently, in local gossiping, there is little overhead on the transmitted protocol information. The hybrid method,
which combines beacons and heartbeats, performs in between. In the bootstrap phase, it resembles more the flooding-based method. This is because during this phase the hierarchy changes, and thus, beacon messages dominate over heartbeat messages.

Hierarchy Maintenance

After the hierarchy has been bootstrapped, a routing protocol continues to maintain it during the whole system lifetime. Such maintenance generates traffic, which again depends on the hierarchy information propagation method, as depicted in Figure 4.13.

![Figure 4.13](image)

Figure 4.13: The stable-state maintenance traffic (TOSSIM). The figure depicts the average (over 10 independent runs) number of messages per round (left) and bytes per message (right) generated by each node in a stable network, depending on the employed hierarchy information propagation method. Essentially, the stable traffic matches the bootstrap traffic with the differences between different methods being more pronounced (cf. Figure 4.12). The only exception is the Hybrid method, which in a stable phase generates lower traffic than in the bootstrap phase. This is because in the stable phase heartbeat messages dominate over beacon messages, and thus, Hybrid is more similar to Gossip.

During maintenance, like during bootstrap, local gossiping (Gossip) generates the fewest and the longest messages, while multi-level scoped flooding (Flood) — the most and the shortest messages. For instance, the difference in the number of messages sent per node per round between Flood[c] and Gossip[s] is nearly a factor of 18. Finally, since in the stable phase, unlike in the bootstrap phase, the Hybrid method generates only heartbeat messages, it performs similarly to local gossiping (Gossip).

The maintenance traffic is necessary for detecting and repairing failures in the hierarchy. To measure the failure recovery latency, the following
micro-benchmarks were conducted. For each method, after the hierarchy had been bootstrapped, a single node was killed to measure the time to recover the hierarchy. Recovery corresponded to a state in which neither the routing address nor the routing table of any alive node contained the identifier of the failed node (i.e., the information about the failed node had been completely removed from the network), and all the alive nodes could successfully route to each other. Afterward, the dead node was reincarnated and was allowed to fully rejoin the system (the rejoining latency was small and is thus omitted in the results). The above steps were then repeated for all other nodes in the network. Figure 4.14 presents the average results depending on the level of a failed node as a cluster head.

\[ h: \text{LH}; N: 1024; \rho: \text{sparse}; p: \text{random} \]

Figure 4.14: The hierarchy recovery latency (TOSSIM). The figure depicts the average number of rounds necessary for recovering node routing addresses and routing tables after a failure of a cluster head, depending on the level of the head in the hierarchy. Essentially, the methods that generate the heaviest traffic (Flood[c]) recover the hierarchy fastest. It is vital to note that these results present the total recovery time of all nodes affected by a failure of a level-i cluster head, which is much longer than the average recovery time of a node. Moreover, as most nodes are only level-0 cluster heads, a failure of a random node requires few and local-only repair activities. In other words, hierarchical routing is relatively robust.

Multi-level scoped flooding with beacons issued in every round irrespective of the level of the issuer (Flood[c]) performs best. Since in this method, each routing entry is refreshed in every round, detecting a cluster head failure is fast irrespective of the level of the head (left plot). In the experiments, an entry was considered dead if it had not been refreshed for 4 consecutive rounds. Moreover, since Flood[c] constructs the hierarchy fast (cf. Figure 4.11) and since address synthesis and recovery in the hierarchical routing library use the same mechanisms (cf. Section 4.2.2), Flood[c] recovers node routing addresses most
quickly as well (right plot). In contrast, in the Gossip and Hybrid methods, detecting a failure of a cluster head is proportional to the distance to the head. Hence, these methods repair node routing tables and labels more slowly. Hybrid, however, is more efficient than Gossip in label recovery (right plot), because it constructs the hierarchy much faster (cf. Figure 4.11). Finally, in multi-level scoped flooding with exponential inter-beacon interval (Flood[e]), the time to detect a cluster head failure is proportional in a landmark hierarchy to the advertisement radius of the head, $R(i)$, which for most of the nodes is longer than the actual distance to the head. Consequently, Flood[e] is the slowest method.

It is vital to note that these results present the total recovery time of all nodes affected by a failure of a level-$i$ cluster head, which is much longer than the average recovery time of a node. Moreover, as most nodes are only level-0 cluster heads, a failure of a random node requires few and local-only repair activities. With some redundancy in the next-hop candidates for routing entries, routing is virtually undisturbed by a failure of a level-0 head. In other words, in addition to offering small state and small stretch, hierarchical routing can be robust in large realistic sensornets.

### 4.3.5. Comparison of Information Propagation Methods

The three evaluated methods for propagating hierarchy information have different characteristics. For a given round length, multi-level scoped flooding with a one-round inter-beacon interval (Flood[c]) bootstraps and recovers the hierarchy fastest, but uses myriads of short messages. In contrast, local gossiping (Gossip) generates the lowest traffic with the lowest overhead on transmitted protocol data, but it takes time to construct and recover the hierarchy. The novel Hybrid approach is in between. Metrics employed to systematically compare these methods thus have to take all such differences into account.

One such a metric suitable for comparing the different hierarchy information propagation methods is the energy consumed in each of the methods. As energy is often a scarce resource in sensornets, many sensornet systems aim to minimize their energy consumption while optimizing other metrics, for instance, the latency of bootstrapping and recovering after failures. Consequently, to get insight into the differences between the three methods for propagating hierarchy information, the energy they consume due to radio activity is compared with robustness they obtain with such energy consumption, for example, in terms of the latencies when bootstrapping a cluster hierarchy.

However, comparing two protocols in terms of energy consumption is far from trivial. There are many factors that affect the energy consumed
by a protocol, such as the hardware components in a particular sensor node platform or the employed medium access control (MAC) layer. In effect, a protocol that consumes little energy in one architecture can become inefficient when used in another architecture. Consequently, rather that proving which of the three methods for propagating hierarchy information is more energy efficient in general, let us highlight the differences in their energy profiles on a sample common hardware and software platform and using the above sample experimental settings.

To this end, as the experimental hardware platform, a common sensor node platform, TelosB [106], is assumed. TelosB is widely used for both research and commercial purposes. Consequently, there is plenty of experimental data on the energy consumed by the radio of this platform in different states [67, 106].

As the software platform, the standard TinyOS 2.0 distribution with the default MAC layer based on low-power listening (LPL) [28, 104] is assumed. In LPL, a node keeps its radio off most of the time. Periodically, however, it turns the radio on just long enough to detect a carrier on the channel. If the node detects a carrier, it leaves the radio on long enough to receive a message. Because the inter-check period is much longer than the message payload transmission time, to give a receiver a chance to hear a message, a transmitter must precede the message with a sufficiently long preamble. More specifically, the total transmission time must exceed the inter-check period. The inter-check period is thus a configurable parameter of LPL that determines how often the radio of a node is turned on to check for a possible transmission and how long the message preamble is. In other words, it corresponds to the trade-off between energy savings on idle radio listening and additional energy expenditures during message transmissions and receptions, which is inherent in all energy-saving MAC layers.

The default TinyOS MAC layer with LPL has well-recognized analytical models of energy consumption [28, 67, 104]. Moreover, it has been widely used again for both research and real-world deployments. Finally, when broadcast traffic is considered, like in the evaluated protocols, that MAC layer works exactly the same as the MAC layer of the proposed Internet architecture for sensornets.

Under the above assumptions, the energy consumed by a radio in each of the hierarchy information propagation methods can be measured by simply combining radio activity traces from the above experiments with publicly available measurements of the current drawn by the radio of a TelosB node [67]. Figure 4.15 depicts the energy consumption of those two of the considered methods that have extreme traffic patterns: local gossiping with multi-frame heartbeat messages sent after a single preamble ($Gossip[s]$) and multi-level scoped flooding with constant inter-beacon issuing interval ($Flood[c]$).
4.3. Low-Level Implementation-Based Simulation

Figure 4.15: The energy consumed with low-power listening (TOSSIM). Low-power listening (LPL) introduces one configuration parameter: the LPL check period. This parameter corresponds to the trade-off between energy savings on idle radio listening and additional energy expenditures during message transmissions and receptions. For each traffic pattern, there is one optimal global setting of the parameter. The figure illustrates the energy consumed during hierarchy bootstrap by different hierarchy information propagation methods configured with the round length $T = 5$ minutes and different values of the LPL check period (the initial period in which link quality estimates are calculated is not taken into account). Gossip[s] consumes less energy than Flood[c] for all reasonable settings of the LPL check period. This is because the traffic pattern of Gossip[s] incurs potentially lower energy overhead on the transmitted information.

The figure illustrates that for the same round length, local gossiping (Gossip[s]) consumes less energy than multi-level scoped flooding with a constant inter-beacon interval (Flood[c]) for all reasonable settings of the LPL check period. In particular, with the LPL check period configured optimally for each of the methods, Gossip[s] requires approximately 3.4 times less energy per round than Flood[c]. This is because even though a node in the two methods transmits a comparable amount of useful information per round, the traffic pattern used to carry that information in Gossip[s] requires less energy compared to Flood[c]: Gossip[s] transmits fewer albeit longer messages than Flood[c], and thus, it incurs potentially lower energy overhead on the transmitted information.

Although Figure 4.15 illustrates which of the two hierarchy information propagation methods requires less energy per time period, it does not show which of the methods uses its energy more efficiently. More specifically, while
Gossip[$s$] uses less energy per round than Flood[$c$], it requires more rounds than Flood[$c$] to bootstrap the hierarchy (cf. Figure 4.11). Therefore, to systematically compare the energy efficiency of these two methods, one has to compare their energy consumption and, for instance, their bootstrap latency under different configurations of the round length. Such a comparison is presented in Figure 4.16.

![Figure 4.16: The energy efficiency of different hierarchy information propagation methods (TOSSIM). The figure compares two extreme methods of hierarchy information propagation, configured with different values of the round length, $T$ (measured in minutes), and the optimal LPL check period for each $T$. The methods are compared with respect to the energy they consume per hour in the hierarchy bootstrap phase and the duration of that phase, excluding the convergence of the link quality estimates. The points in the plot represent the average over 10 independent runs. The figure suggests that, in the adopted software-hardware architecture and the employed experimental settings, Gossip[$s$] consistently outperforms Flood[$c$], thereby, being closer to the ideal hierarchy information propagation method.](image)

The figure suggests that, in the adopted software-hardware architecture and the employed experimental settings, local gossiping ($Gossip[s]$) consistently outperforms multi-level scoped flooding ($Flood[c]$); thereby, it gets closer to the ideal hierarchy information propagation method. For example, when the round length for each of the methods is configured such that both methods consume the same amount of energy per hour, in local gossiping the hierarchy is bootstrapped more than 3 times faster. When the two methods are configured to bootstrap the hierarchy with the same speed, in turn, local gossiping consumes less than a half of the energy consumed by multi-level scoped flooding. Moreover, it is possible...
to configure local gossiping to outperform multi-level scoped flooding in both the metrics.

While the results are consistent with prior and parallel research work [46, 47, 89], it should be noted again that they may be different for different software-hardware architectures or deployment settings. For example, denser networks make the above differences in the energy efficiency of various methods more pronounced. In contrast, some MAC layers, such as those with scheduled channel polling [142], can alleviate the differences in energy consumption. Therefore, in general, a method of propagating hierarchy information should be selected based on its traffic pattern and application requirements rather than based on the energy-related experimental results.

In this view, multi-level scoped flooding with constant inter-beacon interval (Flood[c]) bootstraps and recovers the hierarchy fastest, but uses myriads of short messages. As such, it is most suitable for applications in which heavy traffic is less important than quick construction and recovery of the node routing state.

In contrast, local gossiping (Gossip) generates the lowest traffic with the lowest overhead on transmitted protocol data, but it takes time to construct and recover the hierarchy. Consequently, it is more appropriate for applications that operate on tighter energy budgets, but can tolerate longer periods of disruption (e.g., delay tolerant systems). Alternatively, gossiping may be used in combination with unicast routing, like in the Firecracker [87] and Starburst SSD [4] protocols, which may improve the latency of bootstrapping and recovering the hierarchy in low-latency applications, and thus, constitutes a promising avenue for future improvements.

The Hybrid approach may be a good alternative for both these methods. It offers fast hierarchy bootstrap, like Flood[c], uses mostly low-overhead traffic, like Gossip, and recovers after failures relatively fast. Moreover, the only issue that slows recovery in Hybrid, as compared to Flood[c], is the slow failure detection mechanism inherited from Gossip, in which the number of rounds to detect a failure of a cluster head is proportional to the distance to the head. In some applications, however, failure detection can be improved with an ICMP-like routing-oriented mechanisms. If hierarchical packet forwarding encounters a routing error, it can return a type-3 ICMP message (“Destination Unreachable”) to the source, which then marks a given cluster as failed, yielding almost immediate failure detection.

Finally, multi-level scoped flooding with the exponential beacon issuing pattern (Flood[e]) in practice offers neither fast bootstrap and recovery nor efficient traffic. Thus, this technique is unappealing for real-world applications. Yet, surprisingly many proposed hierarchical routing protocols are based on this
This again illustrates the importance of evaluating sensornet routing protocols in realistic environments.

All in all, not only do the above experiments illustrate that self-managed hierarchical routing can be robust, but also show that one can customize the mechanisms used to ensure robustness. Depending on which aspects of robustness are the most important for a particular application, one can choose those mechanisms from the available design space that address these aspects. Moreover, the design space offers additional solutions, which enable further improving the robustness of hierarchical routing, for example, by reducing the failure detection latencies. Consequently, when also robustness is considered, hierarchical routing is an appealing technique for production-quality routing protocols.

4.4. TESTBED EXPERIMENTS

Hitherto, using low-level simulation it was demonstrated that, in realistic sensornet topologies, hierarchical routing can offer small routing state, small routing stretch, robustness, and self-management capabilities. To confirm that this routing technique can truly operate in the real world, the high- and low-level simulation experiments are complemented with real-world experiments on an actual sensornet testbed at my university.

The architecture and the properties of the testbed can be found in Appendix B. In short, the testbed consists of 60 TelosB nodes in six office rooms. In all experiments, at least 55 of the nodes were operational, that is, the studied node populations had moderate sizes. A TelosB node is a common sensornet hardware platform and incorporates an 8-Mhz 16-bit MSP430 MCU, 10 KB of RAM, 48 KB of flash memory for program code, and a 250-Kbit/s CC2420 IEEE 802.15.4 radio. Therefore, TelosB is a good representative of a resource-constrained sensor node. Depending on the experiment, the protocol round length was varied from 30 seconds to 5 minutes. Likewise, the LPL check period was set globally to a value from 0 ms (nodes always kept their radios on) to 256 ms (in the absence of traffic, nodes turned their radios on just for a few milliseconds 4 times per second). In general, the round length and the LPL check period were configured to minimize contention. In all experiments, the radio transmission power was set to $-15 \text{ dBm}$. As a result, the network diameter oscillated between 4 and 5 hops and the node density was highly heterogeneous: from 8 to 34 neighbors per node. In addition, there were many asymmetric links and some noise during office hours.
4.4. Testbed Experiments

4.4.1. Micro-Benchmarks

The testbed experiments were being conducted for a long time. Many of those experiments were various micro-benchmarks of different design points, phenomena, and failure scenarios. For the sake of brevity, only an illustrative subset of those experiments is presented.

Figure 4.17, for instance, depicts a sample area hierarchy into which the testbed nodes self-organized during one of those experiments. The area hierarchy satisfies the four sample hierarchy properties defined for PL-Gossip in the previous chapter. It consists of three levels. At level 0, it has 55 singleton clusters that correspond to the testbed nodes. At level 1, it has 4 clusters; the heads of two of the clusters happen to be in the same office room. Finally, level 2 is the top level, hence, according to the four properties, it has only one cluster that contains all nodes.

Likewise, Figure 4.18 presents a sample landmark hierarchy. The hierarchy has four levels. There is 1 level-3 landmark (cluster head), 1 level-2 cluster head, 5 level-1 cluster heads, and 48 level-0 cluster heads. The cluster heads are relatively evenly dispersed across the office rooms. Nevertheless, due to the probabilistic nature of the employed cluster head promotion heuristics, there is some redundancy in their number.

In general, the results from the micro-benchmark experiments are consistent with the results from TOSSIM. For example, Table 4.3 summarizes the routing state and stretch values obtained in the experiments with landmark hierarchies. Table 4.4, in turn, illustrates differences between selected methods of propagating hierarchy information in such hierarchies. The numbers in these tables are consistent with the earlier TOSSIM results for 64-node networks. There are some small differences, for example, in the routing tables sizes and the bootstrap latency, but such differences can be attributed to the differences in the experimental settings between TOSSIM and the testbed: the smaller size of the testbed, in the case of routing table sizes, and the fact that all the nodes were not started simultaneously, in the case of the bootstrap latency. Therefore, the testbed experiments suggest that TOSSIM can predict the real-world behavior of the hierarchical routing library well.

4.4.2. Long-Term Network Dynamics

Apart from the micro-benchmarks, also long-lasting testbed experiments with the hierarchical routing library were conducted. The data collected in those experiments illustrate the long-term behavior of a routing protocol and even contain unusual events. As such, they provide noteworthy information on the
Figure 4.17: A sample area hierarchy (testbed). The area hierarchy satisfies the four sample hierarchy properties defined for PL-Gossip in the previous chapter. It consists of three levels. At level 0, it has 55 singleton clusters that correspond to the testbed nodes. At level 1, it has 4 clusters; the heads of two of the clusters happen to be in the same office room. Finally, level 2 is the top level, hence, according to the four properties, it has only one cluster that contains all nodes.

real-world performance of hierarchical routing, in particular, and point-to-point routing, in general.

An important class of real-world phenomena that are not modeled by TOSSIM are dynamic, random interactions of a network with the surrounding environment. Examples of such interactions include wireless noise generated by people’s WiFi-enabled laptops and changes in signal propagation due to mobility in the surrounding environment. These types of interactions make the internode
4.4. Testbed Experiments

Figure 4.18: A sample landmark hierarchy (testbed). The arrows depict the child-parent relationship between landmarks: an arrow from a level-i landmark to a level-i+1 landmark indicates that the lower-level landmark is bound to the higher-level landmark. The hierarchy has four levels, which was sufficient for the employed cluster scaling function $R(i) = 2^i$. There is 1 level-3 landmark (cluster head), 1 level-2 cluster head, 5 level-1 cluster heads, and 48 level-0 cluster heads. The higher-level cluster heads are relatively evenly dispersed across the office rooms.

connectivity dynamic and, thereby, impact the performance of a routing protocol. Wireless noise, for instance, can make some wireless links bursty: such links display short periods of perfect or null packet reception. Mobility, in turn, like repositioning office furniture by just half a meter, can change wireless links considerably and more permanently.

Such dynamic changes affect the performance of a routing protocol. Firstly,
### Table 4.3: The routing state and routing stretch (testbed).
The metrics regard landmark hierarchies built on the testbed; they are similar for area hierarchies. The table contains ranges rather than completely aggregated values to illustrate how the values differ across different runs. The above testbed results are largely consistent with the earlier TOSSIM results for 64-node networks, but there are some small differences. In particular, the routing tables are smaller on the testbed than in TOSSIM. Such differences, however, can be explained by the smaller scale of the testbed as compared to even the smallest, 64-node networks in TOSSIM. Therefore, overall TOSSIM can predict the considered metrics relatively well.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Average</th>
<th>99th Percentile = Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>routing table size</td>
<td>4.95–9.71</td>
<td>7–14</td>
</tr>
<tr>
<td>hop stretch</td>
<td>1.00–1.05</td>
<td>1.33–2.66</td>
</tr>
<tr>
<td>neighborhood size</td>
<td>19.51–23.65</td>
<td>26–34</td>
</tr>
</tbody>
</table>

### Table 4.4: Comparison of different hierarchy information propagation methods (testbed).
The metrics regard landmark hierarchies built on the testbed. Again, these results are largely consistent with the TOSSIM results for 64-node networks with a few small differences. For example, because all testbed nodes were not started simultaneously, as in TOSSIM, the bootstrap time values are larger in the testbed than in TOSSIM. By and large, however, TOSSIM results match the testbed results also with respect to the metrics describing the dynamic behavior of hierarchical routing.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Bootstrap Time [Rounds]</th>
<th>Stable-State Messages Per Node Per Round</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flooded</td>
<td>24</td>
<td>2.07413</td>
</tr>
<tr>
<td>Gossip/ms</td>
<td>19</td>
<td>1.00000</td>
</tr>
<tr>
<td>Hybrid/ms</td>
<td>10</td>
<td>1.00036</td>
</tr>
</tbody>
</table>

they can impair the message delivery rates and the transmission stretch. To alleviate these negative effects, the routing library implements some of the mechanisms enumerated in the previous chapter. These mechanisms have been shown to effectively improve the message delivery rates and reduce the transmission stretch. Secondly, the changes in connectivity may result in the changes of node labels. Since the label of a node is the routing address, a change in the label disrupts the application on top of the routing protocol. Therefore, applications employing hierarchical routing and, equally important, address resolution protocols for hierarchical routing have to be designed to anticipate such changes and to recover from them. For example, in some applications, a simple support for recovery after address changes would involve statically designating a few (special) nodes as the keepers of the node label assignments. The keepers would also act as top-level cluster heads, so that they could always be reached by all nodes. When a source node receives a “Destination Unreachable” message, it can contact one of the keepers to verify the destination label.
A sample two-and-a-half-day run of the routing library depicted in Figure 4.19 demonstrates how the aforementioned phenomena impact the performance of a hierarchical routing protocol. The pairwise routing reachability between nodes is lower and more variable during working hours than during nights. During the first day (from 8:00 AM to 8:00 PM), 6 nodes changed their labels as the result of connectivity changes, amounting to 16 label changes in total during that day. In contrast, there were no label changes during the subsequent night. Those connectivity changes during working hours resulted most likely from the aforementioned noise and mobility in the testbed surroundings. There was also an emergency event during the second day, which resulted in the whole building being evacuated. Although this cannot be verified, it is likely that that event also led to a drop in the internode reachability in the afternoon of the second day, most probably because a crowd of people with their laptops on storming through a corridor with the testbed office rooms may disrupt connectivity.

![Figure 4.19](image.png)

**Figure 4.19:** A sample evolution of the pairwise internode reachability (testbed). For the plot it is assumed that node A can reach node B if and only if A can successfully route to B. The pairwise reachability is the fraction of node pairs that can reach each other. The experiment involved a landmark hierarchy which was maintained using the hybrid information propagation method. The figure illustrates how different phenomena impact the performance of a hierarchical routing protocol.

Another incident experienced during the long-term experiments had more grave consequences. In one protocol run, depicted in Figure 4.20, after a level-1 cluster had been created, the link quality estimates started to randomly fluctuate from 0% to 100% packet reception rate, even for links that were normally stable. As a result, more and more cluster heads started to falsely detect that their parent clusters were no longer reachable. In effect, more and more higher-level clusters were created which further exacerbated the situation. Eventually, the network collapsed.

A long, in-depth investigation of that phenomenon revealed that the failure
Figure 4.20: A Byzantine evolution of the internode reachability (testbed). The figure depicts Byzantine behavior caused by a bug in the compiler for the employed sensor nodes. The experiment involved a landmark hierarchy which was maintained using the multi-level scoped flooding with constant inter-beacon interval information propagation method. At some point after a level-1 cluster had been created, the bug caused the link quality estimates to randomly fluctuate from 0% to 100% packet reception rate, hence the drop of the pairwise node reachability after the initial peak. As a result, more and more cluster heads started to falsely detect that their parent clusters are no longer reachable. In effect, more and more higher-level clusters were created, hence the growth of the median level as cluster head (in a normal network, it is equal to 0, as most nodes are only level-0 cluster heads). The constant creation of new higher-level cluster heads further exacerbated the situation, hence the continuous drop of the pairwise node reachability and the continuous increase of the median level of a node as cluster head. In the end, the network collapsed.

was actually caused by a bug in the compiler for the MSP430 MCU employed in TelosB sensor nodes. The bug caused the compiler to incorrectly handle 16-bit multiplication by 255, which is used in the link quality estimation code. Another group discovered the bug simultaneously, which spawned a considerable discussion in the TinyOS development forum.

While the bug was a Byzantine one, and, arguably, it should not occur in the real world, it illustrates a potential limitation of PL-Gossip and the hierarchical routing library. More specifically, neither of these two involves heuristics for cluster rebalancing, that is, dissolving a cluster when it has too few subclusters. As a result, when there are no node failures, but only disruptive connectivity changes (i.e., ones that require hierarchy changes), the number of higher level clusters, and thus, the node routing state, cannot shrink, but can only grow.

Investigating whether this limitation is relevant in the real world most likely requires employing PL-Gossip or the hierarchical routing library in a prototype of some actual system. If the limitation turns out to be irrelevant in practice, in many applications one may engineer a simple solution for the sake of safety: resetting
the routing state of all or some nodes on demand, so that they can rebuild a (part of a) cluster hierarchy. If in turn the limitation turns out to be problematic, the hierarchy maintenance mechanisms introduced by *PL-Gossip* and implemented in the hierarchical routing library will have to be augmented with cluster rebalancing heuristics. Such modifications, however, would then be necessary also for other small-state routing techniques that use clustering, such as compact routing and some graph embedding techniques. In the experiments, apart from the above, evidently Byzantine behavior, the limitation was not a serious problem.

Therefore, all in all, if Byzantine failures do not occur, the implementation of hierarchical routing provided by the library is relatively robust. In particular, during the worst disruptions in Figure 4.19 more than 84% of all \(N \times (N-1)\) routing paths were valid; this was also the case in other long-term tests. In addition, the network was resilient to massive failures and typically required few rounds to recover. Moreover, similar routing success rates were reported by other sensornet point-to-point routing protocols. These results constitute a strong evidence that, in addition to providing small routing state and small routing stretch, self-managed hierarchical routing can be made robust and reliable in the real world.

4.5. CONCLUDING DISCUSSION

The experimental results presented in this chapter confirm the main thesis: hierarchical routing is a compelling point-to-point routing technique for large sensornets. Essentially, hierarchical routing satisfies all the requirements for such a technique, which were introduced in Section 1.3 and reinforced in Chapter 2. It ensures small routing state that can be a polylogarithmic function of the total node population size, which is essential for scalability and efficiency. Despite the small state, it provides small routing stretch in sensornets, which is possible due to the geometric nature of sensornet deployments and which is important for efficiency and reliability. Finally, it can be made robust to packet loss and changes in the node population and internode connectivity, which is necessary for reliability and scalability. I believe that, together with earlier analytical results, the experimental results constitute strong evidence in favor of using hierarchical routing in sensornets.

Furthermore, the *PL-Gossip* protocol and its development into the TinyOS hierarchical routing library, which enabled conducting the experiments, prove that hierarchical routing can be effectively implemented on today’s sensor node platforms. The implementation can be completely self-managed, which
can reduce the deployment and maintenance costs of a system employing hierarchical routing. This means that hierarchical routing is no longer only theoretically feasible in sensornets. Conversely, it is an appealing technique for practical production-quality routing solutions, and the protocols presented in this dissertation can be a good starting point for such solutions.

Such production-quality solutions differ from the presented research-oriented ones in that in a production-quality environment a protocol has to work as a part of a system and has to abide by some standards. Consequently, employing hierarchical routing in production-quality systems requires addressing some additional issues, the most important of which were enumerated while discussing the results. Examples of such issues include node address changes during network lifetime, limited memory pools for node link tables, high failure detection latencies, and, possibly, a lack of cluster dissolution heuristics.

However, I believe that these issues can be addressed efficiently. In particular, it is likely that the simple, practical solutions given when describing the issues can be adopted by many applications. Moreover, such issues are also inherent in other small-state routing techniques, which were surveyed in Section 2.3 Consequently, to use small-state routing in production-quality sensornets, solutions to those issues have to be developed anyway. Therefore, taking everything into consideration, hierarchical routing is indeed a compelling point-to-point routing technique for large sensornets.
Chapter 5
Comparing Routing Techniques Spectrum

In the two previous chapters, it was demonstrated how to enable hierarchical routing in sensornets and shown that hierarchical routing is a compelling point-to-point routing technique for sensornets. More specifically, it was shown that, in practice, hierarchical routing can simultaneously offer small routing state, small routing stretch, robustness, and self-management capabilities. That work thus fills the gap in the experimental work on the routing techniques spectrum, which was visualized in Figure 2.3 on page 62. Equally important, however, the work from the previous chapters enables experimentally comparing the entire routing techniques spectrum.

Therefore, to complete this dissertation, this chapter deals with such comparisons; two main contributions are made. First, to enable conducting systematic comparisons of different routing techniques, together with Tahir Azim, who was responsible for the TinyOS 2.0 implementation of compact

1Tahir Azim, Ph.D. student, Department of Computer Science, Stanford University, 284 Gates

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission of the author and/or a fee.

Copyright © 2010 by Konrad Stanislaw Iwanicki.
routing and beacon vector routing, we have designed and implemented a point-to-point routing library for sensornets. The library contains uniform implementations of four routing techniques that together represent the entire state-stretch spectrum: shortest-path routing, compact routing, hierarchical routing, and beacon vector routing (constant-state routing). Second, using the library, we have conducted an unprecedented experimental comparison of the routing techniques spectrum. The experiments were conducted on three platforms: in TOSSIM 2.1, a low-level simulator for TinyOS, on KonTest, the 60-node testbed at Vrije Universiteit Amsterdam (see Appendix B), and on MoteLab, a 190-node testbed at Harvard University.

The obtained experimental results illustrate the differences in the performance of different routing techniques from the state-stretch trade-off perspective; in particular, they reinforce the thesis formulated for this dissertation. Therefore, the results can guide the initial choice of potential routing techniques for a given sensornet system. The library itself, in turn, can facilitate the choice of a particular technique. Not only can it enable systematic on-demand comparisons of the techniques spectrum in one’s own settings, including studies of metrics beyond state and stretch, but can also facilitate developing novel solutions for different aspects of routing, thereby fostering innovation.

The rest of this chapter is organized as follows. Section 5.1 starts by motivating the research presented in this chapter. Section 5.2 and 5.3 describe the techniques selected for and the architecture of the aforementioned routing library. Section 5.4 and 5.5 by discussing the initial experimental results obtained with the library, illustrate how different techniques from the state-stretch trade-off spectrum compare with each other. Finally, Section 5.6 concludes by summarizing the merits and drawbacks different routing techniques. An overview of the library sources can be found in Appendix C.2.

5.1. MOTIVATION

Given the spectrum of routing techniques, which was discussed in Section 2.3 choosing a technique that is best suited for a particular application may be challenging. While theory explains asymptotic bounds on the state and stretch of different techniques, it provides limited information on their practical performance. In particular, as shown for hierarchical routing in the previous chapter, the practical performance of some techniques does not approach the
worst-case state/stretch bounds in the “geometric” topologies of sensornets; conversely, the techniques perform well in such topologies. In contrast to theory, existing experimental results do provide information on the practical performance. However, due to differences in the experimental settings they were obtained in, the extent to which they can be compared is severely limited: it is difficult to reason which techniques would perform better in a particular deployment. For these reasons, being responsible for different competing routing techniques, we wanted to experimentally compare the performance of those techniques in a number of common experimental settings.

However, the existing sensornet implementations of the techniques are typically monolithic libraries combining several different pieces of functionality. Such functionality often exceeds beyond what is traditionally considered a routing protocol, usually incorporating parts of lower or higher layers in the routing protocol code, such as node address resolution. In other cases, the functionality covers only some aspects of routing, such as topology maintenance and next hop lookups, leaving the implementation of others to the user. Moreover, the existing implementations usually adopt customized, integrated solutions for various aspects of routing, including link quality estimation, topology maintenance, and packet forwarding. As a result, without in-depth knowledge about all design decisions within a given implementation, it is difficult to disqualify a particular experimental result as an artifact of some implementation-specific solution adopted for some aspect of routing. By and large, nonuniform monolithic implementations make conducting and interpreting experimental comparisons challenging at best; in the worst case, wrong conclusions may be drawn from such comparisons.

Therefore, to cope with these problems, we decided to re-implement selected routing techniques. As a result, we have developed a point-to-point routing library for TinyOS 2.1. The library contains implementations of four routing techniques, which, from the state-stretch trade-off perspective, together represent the entire spectrum (cf. Section 2.3): shortest-path routing (SPR), which requires a linear state and offers the minimal stretch, compact routing (CR), which requires a square-root state and offers a small maximal stretch of three, hierarchical routing (HR), which trades off a low polylogarithmic state for a potentially larger stretch, and constant-state routing with virtual beacon-based coordinates (BVR), which needs a small constant state, but the stretch of which can potentially be large. To implement the techniques in a uniform manner, the library is modular and makes extensive use of decomposition. It decomposes routing protocols into widely recognized abstractions reflecting different aspects of routing (cf. Section 3.1). Each of these abstractions is further decomposed into functionality
that is specific to a particular routing technique and functionality that is common for all considered techniques.

We believe that such an in-depth decomposition enables systematic experimental comparisons of the entire routing techniques spectrum and promotes innovation. First and foremost, it facilitates understanding the performance impact of particular design decisions in various routing abstractions on different routing techniques: since for each routing abstraction, there is a clear separation between code that is common for all routing techniques and code that is technique-specific, it is relatively easy to classify a decision as technique-specific or generic. Furthermore, because the common code constitutes the great majority of the whole library code base, implementations of different techniques inherently make the same design decisions and differ only where necessary, thereby facilitating systematic comparisons. Finally, as the routing abstractions comprising a protocol in our library are well recognized, not only can one test different design decisions within the existing implementations of these abstractions, but can also develop novel, innovative implementations of these abstractions.

Below, first, each of the selected techniques is described, and then, it is shown how by using decomposition such diverse techniques can be implemented with modules that are mostly shared and involve only a small fraction of technique-specific code.

5.2. SELECTED ROUTING TECHNIQUES

We had two major objectives when selecting the routing techniques for the library. First, we wanted to cover each major representative trade-off between routing state and routing stretch, as enumerated in Section 2.3. Second, we preferred techniques that had already been implemented and proved to work on real sensor nodes to the techniques that had been evaluated only with simulation. That led to the choice of the four techniques mentioned above. Although these techniques were described in detail in Section 2.3, they are summarized below for the sake of completeness.

Shortest-Path Routing (SPR): Shortest-path routing represents one end of the state-stretch trade-off spectrum. In this routing technique, the routing address of a node corresponds to the unique identifier of this node. The routing table, in turn, involves one routing entry for every other node, hence in total $O(N)$ entries per node. An entry for a node contains information on the shortest route to that node. Therefore, in shortest-path routing, nodes
can route to each other along the shortest possible paths, which yields the minimal possible stretch, that is, 1. In other words, shortest-path routing requires the maximal routing state, but offers the minimal possible routing stretch in return.

**Compact routing (CR):** Compact routing aims to reduce the routing state while keeping the stretch small. In compact routing, $\sqrt{N}$ nodes are selected as beacons, and each remaining node binds itself to the beacon that is closest in terms of hops. Such a binding determines the routing state of the nodes. More specifically, the routing address of a node is a concatenation of the unique identifier of the node and the identifier of the beacon the node is bound to. The routing table of a node, in turn, consists of two types of entries: first, one entry for every beacon node, and second, one entry for every non-beacon node that is closer to the node than to its own beacon. This organization reduces the number of entries per node to $O(\sqrt{N})$. However, it also increases the routing stretch. If a node has a routing entry for the destination node of a packet, the packet is routed to the destination along the shortest path, as described by the entry; consequently, the stretch is 1. However, if a node has no entry for the destination, the packet is first routed toward the beacon to which the destination node is bound. Only when it arrives at a node that has a routing entry for the destination, is it redirected toward the destination itself. In such a case, the maximal stretch can be up to 3. To sum up, compact routing offers a state that is proportional to the square root of the total node population, and a stretch that does not exceed 3.

**Hierarchical routing (HR):** As described in the previous chapters, hierarchical routing further reduces routing state at the expense of stretch. As a reminder, in hierarchical routing, more specifically, in its variant based on a landmark hierarchy, nodes are organized into an $\mathcal{H} + 1$-level hierarchy of clusters, with one node in each cluster being a cluster head. The routing table of a node involves one routing entry for each level-$i$ cluster head that is within $R(i)$ hops from the node (typically $R(i) = 2^i$). In this way, the number of levels, $H$, and of routing entries can be a polylogarithmic function of the total node population size. At every level, each node selects one cluster head from its routing table and becomes a member of the cluster of that head. In effect, the routing address of the node is a concatenation of the unique identifiers of its cluster heads at subsequent levels. A routed packet is forwarded toward the lowest-level cluster head of the destination node for which the present node has a routing entry. In
the worst case, it is first routed toward the head of the top-level \( \mathcal{H} \) cluster of the destination. As soon as it arrives within \( R(\mathcal{H} - 1) \) hops from the lower-level \( (\mathcal{H} - 1) \) head, it is redirected toward that head, and so on, down to the level-0 head, which represents the destination node itself. Such a redirection process, however, may result in a large, nonconstant maximal routing stretch in some networks. To sum up, hierarchical routing can offer small polylogarithmic routing state, but its worst-case stretch may grow with the node population in some “nongeometric” topologies.

**Constant-state routing with beacon-based coordinates (BVR):**

Constant-state routing further trades off the routing state for stretch, constituting the other end of the state-stretch spectrum. In the selected constant-state routing technique, beacon vector routing (BVR), \( B \) nodes are selected as beacons, where \( B \) is a constant. The routing address of a node is a \( K \)-element vector \( (K \leq B) \), the \( i \)-th element of which denotes the number of hops from the node to the \( i \)-th closest beacon. The routing table contains one entry for each beacon node and one entry for each neighbor with a \( B \)-dimensional vector representing the distance from the neighbor to each of the beacons, in total still \( O(1) \) entries. Routing in BVR is performed greedily by forwarding a packet to the neighbor the \( B \)-dimensional coordinates of which minimize some distance metric to the destination. If greedy forwarding cannot make progress, the packet is forwarded toward the beacon closest to the destination, and greedy forwarding can be resumed whenever it can make progress. If the packet reaches the beacon and greedy forwarding still cannot be resumed, the beacon initiates a scoped flood with the radius equal to the number of hops to the destination, that is, all nodes within that radius from the beacon forward the packet. Therefore, while BVR maintains a constant state, its stretch may be large, especially when scoped flooding is necessary to deliver a packet.

### 5.3. ROUTING TECHNIQUE IMPLEMENTATIONS

To implement the above techniques in a uniform fashion, we analyzed each of them according to the common skeleton for a sensor network point-to-point routing protocol, which was described in detail in Section 3.1. The skeleton identifies three main routing abstractions: packet forwarding, routing state maintenance, and link quality estimation. We decomposed each of the four routing techniques
into these three abstractions. Moreover, for each technique, we decomposed each of the routing abstractions of this technique into functionality that is specific to the technique and functionality that is common for all the considered techniques.

The technique-specific parts turned out to be small; consequently, we implemented them as hook functions invoked from the common code and grouped them together into a single TinyOS module per routing technique. By also adopting opaque data types in the common code (e.g., for representing routing addresses), we made the interfaces of each routing abstraction independent of a particular routing technique. The overall effect is that by switching just one preprocessor definition, the user of our library can switch between the four different routing techniques, potentially without changing the application code. The decomposition process is summarized below.

5.3.1. Link Quality Estimation

As a reminder, the goal of link quality estimation is discovering which neighbors of a node form links with the node such that those links display high packet reception rates in both directions. By keeping estimates of the quality of the wireless links, a routing protocol can choose high-quality links for routing paths and can dynamically change these paths when the quality of some links deteriorates or improves. This increases per-hop packet delivery rates and can decrease the routing stretch.

While most existing routing protocol implementations contain custom link-estimation components, research to date has demonstrated how a link-estimation component should interact with the components corresponding to the other routing abstractions and has yielded implementations of a few link estimation algorithms. Such algorithms typically do not depend on a particular routing technique. Consequently, in our library, link estimation does not currently involve any technique-specific code. Instead, we adopted passive link estimator interfaces, like in PL-Gossip, which just require access to the whole broadcast and unicast traffic passing through a node. More specifically, we slightly modified existing passive link estimators, including a beacon-based estimator [138] and a four-bit link estimator [34], to unify their interfaces and, to some extent, their semantics. By repeating the same process, the user of the library can likely port other custom estimators to the library.

5.3.2. Routing State Maintenance

Routing state maintenance, that is, the maintenance of the routing tables and, possibly, the routing addresses of nodes, constitutes a foundation of routing and
is often the most intricate component in a routing protocol.

Nodes have to continuously maintain their routing tables to account for changes in the node population and in the quality of wireless links, which occur throughout the network lifetime. In each of the considered routing techniques, the routing table of a node has a seemingly different structure: it is flat in shortest-path routing, involves two types of entries in beacon vector routing and compact routing, or reflects a multi-level hierarchy in hierarchical routing. However, by making a few observations, we unified the routing tables in the selected techniques.

In all the techniques, each routing entry corresponds to a node, which we refer to as a **landmark**. We can also generalize that, in all the techniques, landmarks form a hierarchy, that is, each landmark has a certain **level**: in hierarchical routing this is straightforward; in compact routing and beacon vector routing, beacon nodes are level-1 landmarks and non-beacon nodes are level-0 landmarks; in shortest-path routing, all nodes are level-0 landmarks. Each landmark has to also advertise itself to nodes within a specific distance, denoted as **scope**: in hierarchical routing, the scope depends on the level, $i$, like $R(i) = 2^i$; in shortest-path routing, the scope is infinite as all nodes advertise to all other nodes; in compact routing, the scope is infinite for beacon nodes and equal to the distance to the closest beacon for non-beacon nodes; likewise in beacon vector routing, beacons have an infinite scope, and non-beacons a scope of one hop. Apart from the above fields, in every technique, an entry for a landmark has to contain the number of hops and a set of possible next-hop neighbors on the shortest path to the landmark, as well as a few maintenance fields, such as the last sequence number generated by the landmark and the time-to-live for this number, which were discussed in detail in the previous chapters.

By and large, in the above view, a routing entry has conceptually the same fields for all the selected routing techniques. Consequently, we made the implementation of a routing table in our library shared by the techniques. A technique that does not use some of the fields or can infer them from other fields simply does not need to store them. For example, a routing entry in shortest-path routing does not need the level and the scope fields; in hierarchical routing, in turn, the level is necessary, but the scope is not. Conversely, a technique may need some additional fields, such as an adjacency flag in hierarchical routing or the coordinates in the $B$-dimensional beacon space in beacon vector routing. In this case, such fields constitute a technique-specific part of the routing table and are accessed through hook functions. Overall, the routing table provides a uniform interface encompassing entry lookups by the unique identifiers of the landmarks the entries refer to, iteration over the entries, and serialization of the
5.3. Routing Technique Implementations

With such a unified implementation of a routing table, we could also unify the maintenance code for the node tables. There are several ways of maintaining node routing tables, the most common two being flooding and gossiping landmark advertisements (one of the design points studied in the previous chapter). We implemented the gossip-based version, much like the one described in Section 3.4.2. In essence, nodes operate in periods. In every period, each node broadcasts its whole routing table to its neighbors. The neighbors that receive the routing entries of the node use these entries to update their own routing tables, following a distance-vector algorithm. When performed continuously, this algorithm allows nodes to gradually build their routing tables and maintain them when some nodes fail or the internode connectivity changes. Since the algorithm is largely independent of a routing technique, its implementation in our library is mostly shared by all techniques. There are just a few hook functions, for example, for influencing decisions about using a given received routing entry to update the own entry of a node.

In addition to maintaining their routing tables, in some applications of some of the selected routing techniques, nodes may also be required to autonomously synthesize and maintain their routing addresses. Section 3.4.3 described in detail the mechanisms PL-Gossip employs to efficiently maintain routing addresses in hierarchical routing. Routing address maintenance mechanisms for the other three selected routing techniques are beyond the scope of this dissertation, though. Nevertheless, in our library, we implemented each routing technique with both static and dynamically maintained addresses. Moreover, it was possible to make the dynamic address maintenance code shared to a large extent by all the techniques. The key observation is that dynamic address maintenance essentially corresponds to promoting some nodes to higher-level landmarks and ensuring that a certain number of such landmarks always exist. In the considered techniques, this can be realized with probabilistic heuristics in which the technique-specific features are just the probability that a node becomes a landmark and the latency between subsequent promotion attempts.

5.3.3. Packet Forwarding

Packet forwarding involves the activities associated with forwarding a packet, from the moment the packet is received by a node until the moment the responsibility for the packet is passed to the next-hop node(s). These activities involve detection of packet duplicates, queuing packets, looking up a list of next-hop candidates, selecting one or more next hops from the candidate
list, transmitting the packet, and possibly recovering if the transmission has failed. As in PL-Gossip, packet forwarding can be largely independent of a particular routing technique, the only technique-specific parts being initializing technique-specific records in a packet header and looking up the list of potential next-hop candidates. In this way, in our library we implemented two variants of packet forwarding, namely unicast forwarding and scoped floods, but we are considering implementing other forwarding methods, such as opportunistic forwarding [113].

Moreover, while we made the whole next hop lookup process completely technique-specific, it could have been shared to some extent. To achieve this, one would just have to make use of our earlier observations regarding the similarity in the routing tables of different techniques. We decided against sharing the next-hop lookup code between the selected techniques for two reasons: first, it would impair the readability of the code, and second, in most techniques the code consists of just a few lines, and thus, it constitutes just a tiny fraction of the whole code base.

5.3.4. Routing Code Breakdown

To illustrate the extent to which the functionality in our library is shared among the techniques and how much the techniques differ, Table 5.1 presents the breakdown of the routing code in the library. The table shows that, as we argued in the above analysis, the great majority of the code is shared by all the selected routing techniques. This ensures that, wherever possible, the implementations of these techniques make the same design decisions. A clear separation between technique-specific and common code, as provided by the hook functions, facilitates distinguishing between technique-specific and common design decisions. This, combined with the decomposition of a routing protocol into standard, widely-recognized routing abstractions, facilitates attributing performance results to particular design decisions, which we believe makes the experimental comparisons of different routing techniques more systematic.

To a limited extent, the table also illustrates the relative difficulty of implementing the selected techniques. For example, shortest-path routing (SPR) requires fewest technique-specific lines of code and from our experience was the easiest to implement. In contrast, beacon vector routing (BVR) involves most technique-specific code and was also the most difficult to implement correctly.
5.4. LOW-LEVEL IMPLEMENTATION-BASED SIMULATION

Our library enables conducting systematic experimental comparisons of the four selected routing techniques that cover the entire state-stretch trade-off spectrum. While we are still conducting experiments on various platforms, in the remainder of this chapter some selected results we have obtained so far are presented. The objective of these experiments is to illustrate how hierarchical routing compares against other point-to-point routing techniques in terms of routing state and routing stretch. To the best of my knowledge, this is the first experimental comparison that involves routing techniques from the entire state-stretch trade-off spectrum.

The first experimental platform for our library was a low-level TinyOS simulator, TOSSIM 2.1. We have been conducting numerous low-level simulation experiments. For the sake of brevity, however, let us focus only on a subset of these experiments, which is sufficient to illustrate general trends in the results and to describe some phenomena we experienced. In short, those experiments were conducted in various network topologies with realistic models of low-power wireless communication, similar to the topologies employed in the experiments from the previous chapter.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Lines of Code (SLOCCount)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Common</td>
</tr>
<tr>
<td>link quality estimation</td>
<td>997</td>
</tr>
<tr>
<td>routing state maintenance</td>
<td>1954</td>
</tr>
<tr>
<td>packet forwarding</td>
<td>1505</td>
</tr>
<tr>
<td>shared by the abstractions</td>
<td>52</td>
</tr>
<tr>
<td>TOTAL</td>
<td>4508</td>
</tr>
</tbody>
</table>

| TECH.-SPEC./ (COMMON + TECH.-SPEC.) | 4.7% | 5.6% | 8.4% | 11.4% |

Table 5.1: The code breakdown of routing protocols with static addresses. The shared code refers to the code that cannot be attributed to a single abstraction, such as the definition of and some operations on a routing address; its volume may be different for different techniques (e.g., the definition of and operations on a routing address are more elaborate in hierarchical routing than in shortest-path routing). The common code always describes the simplest variant of a given piece of functionality, for example, the simplest link estimator, the simplest packet forwarding routines, and so on. Despite this, for any routing technique, the technique-specific code does not exceed 12% of the whole protocol code. In other words, different routing techniques have highly uniform implementations in our library. Moreover, to some extent, the figure also indicates the difficulty of implementing a given routing technique as compared to the other techniques. For instance, even with static addresses, the percentage of technique-specific code is 8.4% for hierarchical routing (HR) and 5.6% for compact routing (CR), which seems to confirm our empirical experience that hierarchical routing is more difficult to implement than compact routing.
More specifically, in those experiments, nodes were deployed randomly in a square area, and we varied their number as well as the size of the area. Using TOSSIM tools, we generated realistic radio gain values for each of the resulting deployments. We also used the Cassino Lab noise traces to generate even more realistic noise models, a feature that became available with TinyOS 2.1 \cite{82}. Like in the previous chapter, to measure link quality, a standard passive beacon-based estimator \cite{138} was employed. As before, only links with at least a 55% bidirectional packet reception rate were considered when selecting routing paths. The routing state maintenance component dynamically synthesized and maintained node routing addresses with probabilistic heuristics, as mentioned in Section \ref{sec:probabilistic-heuristics}. The packet forwarding routines recovered from unacknowledged transmissions by retransmitting packets up to 10 times per hop; the back-off between retransmissions was 1 s to cope with potential link burstiness \cite{119}. In beacon vector routing, scoped floods were also allowed as a fall-back mechanism when unicast forwarding was unable to make progress.

### 5.4.1. Routing State

Figure \ref{fig:network-state} depicts the routing state obtained for the four techniques in networks with exponentially growing node populations. Like previously, the metric for the state is the number of routing entries a node has to maintain. Essentially, the figure illustrates that each technique occupies its respective position in the state dimension of the techniques spectrum, as explained in detail in Section \ref{sec:techniques-spectrum} and summarized in Section \ref{sec:summarized-results}.

In shortest-path routing (SPR), the number of routing entries of a node grows linearly with the total node population size. With a single routing entry requiring at least 10 bytes, such a fast-growing routing state means that in practical deployments shortest-path routing is unlikely to scale to networks beyond some tens of nodes. Even though on the MoteLab testbed we did run shortest-path routing in 100+-node networks, conducting those experiments required compromising on the memory allocated to other services, more specifically, we allocated less memory to packet buffers. Although such a compromise did not affect the experiments, it may be impossible in actual real-world sensornet systems. On a positive side, in shortest-path routing, each node maintains the same amount of entries (compare the top plot with the bottom one), which allows to fully utilize the memory pools provisioned for the entries.

The analytical results regarding the node routing state in compact routing (CR) bound the state to $O(\sqrt{N})$; the results depicted in Figure \ref{fig:network-state} are consistent: compared to shortest-path routing, compact routing offers a significantly smaller
The routing state vs. the network size (TOSSIM). The figure depicts the average and the 99-th-percentile number of routing entries in different routing techniques run in exponentially growing node populations. The figure shows that, from the state perspective, the selected techniques indeed represent the entire spectrum. In particular, there is a large difference in state between the two most competing techniques: compact routing (CR) and hierarchical routing (HR).

Moreover, the figure suggests that, in 99% of the cases, the state of a node does not exceed twice the average state, which means that, when provisioning memory pools for the 99-th percentile, one can expect that on average a node will
utilize 50% of its pool. However, the constant associated with the square root seems to be large, which may limit the scalability in practical deployments. For example, a routing table in the sparse 1024-node network from the figure requires 127 entries on average, which is approximately 4 times the square root of the node population size \(4 \cdot \sqrt{1024} = 128\). All in all, when memory limitations of sensor nodes are considered, compact routing can in practice scale up to some hundreds of nodes.

Hierarchical routing (HR) further reduces routing state, and, more importantly, the pace at which this state grows. The results depicted in Figure 5.1 are consistent with the results from the previous chapter. For example, a routing table in the sparse 1024-node network from the figure requires 43 entries on average, and 63 entries in 99% of the cases. Consequently, from the node memory perspective, hierarchical routing can in practice scale up even to some thousands of nodes. Moreover, the results from the figure were obtained for landmark hierarchies. However, as shown in Section 4.3.3, area hierarchies can offer even better scalability.

Constant-state routing with beacon-based virtual coordinates, according to the beacon vector routing algorithm (BVR), offers a constant state that in addition involves few routing entries. For example, routing tables in the sparse networks from Figure 5.1 consist of approximately 25 entries on average, and 27 entries in 99% of the cases, depending on the number of neighbors of a node. Therefore, BVR should be able to scale potentially infinitely. For various reasons, however, this is not the case, as will be demonstrated shortly.

Figure 5.2 depicts the routing state obtained for the four techniques in networks with varying node densities. In shortest-path routing (SPR), the density of the network does not impact routing state. In all the other techniques, the state is proportional to the network density. Nevertheless, the factor of proportionality may be different for different techniques. In particular, it is smaller for hierarchical routing (HR) and beacon vector routing (BVR) than for compact routing (CR). Such differences in the factor of proportionality, however, can be controlled to some extent by fine-tuning the probabilistic heuristics for landmark promotion. Moreover, if area hierarchies are used instead of landmark hierarchies, hierarchical routing can be optimized such that its state will decrease with the increase in the node density, as explained in Section 4.3.3. The general observation that can be made based on Figure 5.2 is that when the network density increases, the difference in the state maintained by the four routing techniques diminishes.
5.4. Low-Level Implementation-Based Simulation

![Graph](image)

**Figure 5.2:** The routing state vs. the network density (TOSSIM). The figure depicts the average and the 99-th-percentile number of routing entries in different routing techniques run in networks with increasing node densities. In all the techniques, except for shortest-path routing, routing state grows with the node density.

5.4.2. Routing Stretch

According to the state-stretch trade-off in point-to-point routing, the amount of routing state nodes maintain in a routing technique is directly related to the routing stretch this technique can offer. Figure [5.2] presents the routing stretch obtained in the experiments from Figure [5.1].
Figure 5.3: The routing stretch vs. the network size (TOSSIM). The figure depicts the average and the 99-th-percentile routing stretch in different routing techniques run in exponentially growing node populations. The routing stretch is defined here as a product of hop stretch and transmission stretch, hence it can be above 1 for shortest-path routing (SPR). For BVR, numbers are shown, because they are outside the range of the y-axis. Combined with Figure 5.1, this figure shows that the larger the state the smaller the stretch and vice versa. Yet, despite a large reduction in state in hierarchical routing (HR) as compared to compact routing (CR), the stretch in HR is only slightly larger than the one in CR, and is close to the optimal one of SPR. Such small differences in stretch despite large reductions in state are due to the geometric nature of sensornets.
To obtain the stretch results, we randomly selected 100 nodes and made them route packets to each other. In other words, the routing was performed over 100 · 99 source-destination pairs. As we did not want packets from different sources to interfere with each other, only one source at a time was allowed to generate packets. More specifically, the first of the 100 sources generated 10 packets for each of the 99 destinations; the interval between subsequent packets generated by the source was 1 second. After all 990 packets had been routed by the first source, the second source continued the process, and so on. Therefore, in total, nearly 100,000 packets were routed in one experiment.

As explained in Section 4.3.2, two main metrics are typically used to measure routing stretch in low-power wireless networks. First, the hop stretch of a routing path between two nodes is defined as the ratio of the number of hops on this routing path to the number of hops on the shortest possible path between the two nodes in the internode connectivity graph. Second, the transmission stretch of a packet routed over a path between two nodes is defined as the ratio of the number of transmissions to deliver the packet to the number of hops on the path. While in the previous chapter hop stretch and transmission stretch were measured separately, in this chapter they are combined. This is because beacon vector routing (BVR) involves scoped flooding as a fall-back mechanism when unicast forwarding cannot make progress (cf. Section 5.2). In scoped flooding, however, even though a packet may reach a destination in few hops, the number of transmissions to deliver the packet is high: all nodes within the radius of a scoped flood rebroadcast the packet. To make the comparisons of BVR and the other three techniques fair, routing stretch is thus defined as the product of hop stretch and transmission stretch.

An immediate particular observation one can make in Figure 5.3, and which is associated with the above definition of routing stretch is that the routing stretch of shortest-path routing (SPR) is above 1: approximately 1.1 on average and not greater than 2 in 99% of the cases. This is because routing a packet over even the shortest possible path may require a few retransmissions, for example, because the environmental noise may disrupt some transmissions at some intermediate nodes. In other words, even a technique that offers the minimal possible stretch performs slightly worse in practice.

An overall observation when combining Figure 5.3 with Figure 5.1 is that the trade-off between routing state and routing stretch does exist. The larger the state a routing technique requires, the smaller the stretch it can offer, and vice versa. However, the differences in the stretch of different techniques are relatively small.

More specifically, the stretch offered by two selected small-state techniques, namely compact routing (CR) and hierarchical routing (HR), is close to the stretch
of shortest-path routing (SPR). For example, in the sparse 1024-node network in the figure the stretch of hierarchical routing is equal to 1.35 on average and does not exceed 2.33 in 99% of the cases; for compact routing these values are 1.25 and 2, respectively. Again, such a small stretch is a consequence of the embedding of sensornets in physical space. Such embedding results in “geometric” network topologies, which are characterized by large diameters compared to the node population sizes. In such topologies, the state aggregation inherent in small-state routing techniques, does not lead to a large increase in stretch [70].

Such a small difference in stretch between compact routing and hierarchical routing may suggest that, of these two techniques, the one with the smaller state may be more appealing in practical sensornet deployments, especially since the difference in state of the two techniques is large (cf. Figure 5.1). In Section 5.4.1 it was argued that due to a substantially smaller state, hierarchical routing can offer an order of magnitude better scalability than compact routing (thousands of nodes vs. hundreds of nodes). That analysis was based just on the memory constraints of sensor nodes. However, the amount of state in a routing technique affects not only the memory necessary to support this technique, but, arguably more importantly, the bandwidth and energy required to enable routing according to this technique. The volume of maintenance traffic is typically proportional to the amount of state nodes maintain. Therefore, due to a significantly smaller state, hierarchical routing can use less bandwidth and energy to maintain the routing infrastructure, as compared to compact routing. This, in turn, enables prolonging the lifetime of a deployment or improving the latency of recovering after node failures and changes in the internode connectivity (cf. Section 4.3.5). Consequently, all in all, since the difference in stretch between compact routing and hierarchical routing is relatively small while the difference in state is large, hierarchical routing may arguably be more appealing than compact routing in many sensornet deployments.

Finally, as expected from the state-stretch trade-off, due to maintaining only a constant state, beacon vector routing (BVR) offers the largest stretch. However, the stretch values obtained for BVR are excessively large. This is a direct consequence of using scoped flooding as a fall-back mechanism in BVR and will be analyzed in detail shortly. First, however, let us summarize how the stretch of the techniques changes with a change in the node density.

Figure 5.4 presents the routing stretch obtained in the experiments with varying node densities from Figure 5.2. With an increase in the node density, the stretch of most techniques seems to remain constant. This is likely a consequence of the fact that with an increase in the network density, the routing state increases as well (cf. Figure 5.2). In particular, in denser networks compact routing (CR) and
Figure 5.4: The routing stretch vs. the network density (TOSSIM). The figure depicts the average and the 99-th-percentile routing stretch in different routing techniques run in networks with increasing node densities. Due to an increase in state with an increase in the node density, the stretch seems to remain constant. The only exception is beacon vector routing (BVR) in which the state grows significantly (numbers are shown instead of an actual plot). This is because in denser networks the cost of scoped flooding grows, which increases stretch. Moreover, the stretch in BVR appears unstable in denser networks. This is because just 10 beacons in a 1024-node network is apparently insufficient to guarantee a lack of scoped floods when routing between some nodes, which results in a highly variable stretch (cf. Figure 5.3).
hierarchical routing (HR) can offer stretch values that are similarly small as those in sparse networks.

The only technique that seems anomalous when the above stretch results are considered is beacon vector routing (BVR). The excessively large stretch of beacon vector routing is a consequence of using scoped flooding as a fall-back mechanism (cf. Section 5.2). More specifically, BVR appoints \( B \) nodes as beacons to create a \( B \)-dimensional virtual coordinate space, where \( B \) is a constant. Normally, greedy unicast routing is used to forward a packet toward a node that is closer to the destination in this \( B \)-dimensional space than the current forwarding node. In such a case, the stretch is typically small. Sometimes, however, a greedily forwarded packet reaches the beacon closest to the destination and cannot make any further progress, which corresponds to a routing void in geographic routing (cf. Section 2.3.2). In such a case, BVR requires the beacon to perform a scoped flood with a radius equal to the number of hops from the beacon to the destination. In a scoped flood, however, all nodes within the flood radius rebroadcast the packet, which boosts the routing stretch as compared to routing techniques based solely on unicast forwarding (see Figure 5.3). Moreover, the denser the network is, the more nodes take part in a scoped flood with a given radius, which increases the stretch of BVR in denser networks.

The number of scoped floods in BVR can be reduced by increasing the number of beacons, \( B \), as depicted in the left plot of Figure 5.5. Essentially, the more beacons there are in a network, the higher the probability that greedy forwarding can make progress and consequently the lower and more stable the stretch. Increasing the number of beacons, however, increases the number of routing entries each node has to maintain. This provokes a question whether beacon vector routing is indeed a constant-state routing technique or whether its state should depend on the node population size.

Moreover, unlike in the other three techniques, in which the size of a routing entry is constant, in BVR it depends on the number of beacons, \( B \). Therefore, with an increase in \( B \), not only does the number of routing entries grow, but also the size of a single routing entry increases. The overall effect of this dependency on \( B \) is that, when measured in bytes, the node routing state in beacon vector routing can be much larger than the state in hierarchical routing, as depicted in the right plot of Figure 5.5. This is true even when BVR uses as few as 10 beacons. Moreover, in large networks, despite the larger state, the stretch of BVR with 10 beacons is several times larger than the stretch of hierarchical routing. In other words, the trade-off between routing state and stretch in beacon vector routing is far from being an optimal one.

The highly suboptimal state-stretch trade-off is not the only problem of BVR,
5.4. LOW-LEVEL IMPLEMENTATION-BASED SIMULATION

\[ \rho: \text{sparse; } \rho: \text{grid (unit-disk connectivity model)} \]

**Figure 5.5:** The stretch and state of BVR in various configurations (TOSSIM). The figure depicts the average routing stretch and the average routing state measured in bytes for different configurations of beacon vector routing (BVR). The figure essentially shows that although BVR is considered as a constant state routing technique, if its state does not grow with an increase in the node population size, its stretch becomes excessively large. Moreover, even for the smallest considered state configuration (i.e., the one resulting in a large stretch), the state in BVR measured in bytes is larger than the state in hierarchical routing (HR). In other words, the state-stretch trade-off offered by BVR is far from being an optimal one.

though. More specifically, during our experiments, we discovered an inherent flaw in the original beacon vector routing algorithm [35]. Even with scoped floods used as a fall-back mechanisms for unicast forwarding, BVR is unable to guarantee packet delivery. Even though a physical path between two nodes may exist, BVR may not be able to discover such a path. Instead, greedy forwarding toward destination, combined with the heuristic for forwarding toward the beacon closest to the destination, may push routed packets into an infinite routing loop. In other words, the original BVR algorithm may cause permanent routing failures for some node pairs, even in ideal networks with unit-disk connectivity models. We observed such behavior for up to 5% of source-destination pairs, depending on the number of nodes and beacons. Since such behavior is unacceptable for a routing protocol, additional mechanisms for breaking routing loops are necessary. We later discovered hacks implementing such mechanisms in the forwarding code of the existing BVR implementation.

To sum up, the beacon vector routing protocol was an important piece of research, as it was the first one to demonstrate how point-to-point routing can be supported in sensornets. However, its suboptimal state-stretch trade-off and other problems can raise doubts as to whether BVR should be used in any practical deployments, especially since compact routing and hierarchical routing
can perform significantly better.

5.5. TESTBED EXPERIMENTS

Apart from low-level simulation experiments, we have been conducting extensive experiments on various sensornet testbeds. In this section, sample results from two such testbeds are presented, namely from KonTest, a 60-node testbed at Vrije Universiteit Amsterdam (see Appendix B), and from MoteLab, a 190-node testbed deployed at Harvard University.

Although the experimental scenarios run on both those testbeds were largely similar to the ones from TOSSIM, there were some small differences. For example, whereas on KonTest we set the radio transmission power to $-15\,\text{dBm}$, on MoteLab we changed it to $0\,\text{dBm}$, because the nodes on MoteLab are deployed more sparsely. Likewise, due to memory limitations of real sensor nodes, we had to compromise on the memory allocated to various services. More specifically, to accommodate memory pools for the $100^+$ routing entries in shortest-path routing, we had to limit the memory allocated to packet buffers on MoteLab; we had to do the same for beacon vector routing, as the routing state of $BVR$ measured in bytes was also too large for TelosB nodes. Apart from such differences, however, the experimental settings on the testbed were the same as in TOSSIM.

Table 5.2 compares the four routing techniques on the two selected testbeds. The testbeds vary in scale. KonTest encompasses 60 nodes, 53 of which were active during the experiments. MoteLab, in turn, consists of 190 nodes; 119 of those nodes were active during experiments, but since 15 of the active nodes were isolated (i.e., they were unable to communicate with the other nodes), all metrics were measured only among the 104 connected nodes. With the two different settings of the radio transmission power, the network density on both testbeds was similar. The resulting diameters were 4 hops for KonTest and 6 hops for MoteLab. All in all, I believe that the two testbeds constitute representative examples of moderate- and medium-size sensornet deployments.

The table suggests that the performance of the four selected routing techniques in the real world is similar to their performance in TOSSIM. There are again small differences, though.

For example, on KonTest, the routing state of hierarchical routing ($HR$) is significantly smaller than the state of the other techniques. Such a small state can be attributed to the relatively small scale of KonTest and the cross-layer optimization of hierarchical routing discussed in Section 4.3.1; the optimization cannot be applied to the other three techniques, thus their state is significantly
Table 5.2: The performance of different routing techniques (two testbeds). The table illustrates that the routing technique implementations in our library work seamlessly in the real world. Moreover, their performance is largely consistent with the results obtained with low-level simulation. Any small differences may be attributed to the smaller scale of the testbed experiments. MoteLab is marked with * because out of its 119 then active nodes approximately 15 were isolated; consequently, only the 104 connected nodes are considered here.

<table>
<thead>
<tr>
<th>METRIC</th>
<th>KonTest</th>
<th>MoteLab</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPR</td>
<td>CR</td>
</tr>
<tr>
<td>num. active nodes</td>
<td>53</td>
<td>104[119]*</td>
</tr>
<tr>
<td>diameter [hops]</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>avg. neighbors</td>
<td>17.48</td>
<td>14.10</td>
</tr>
<tr>
<td>99-th perc. neighbors</td>
<td>25</td>
<td>27</td>
</tr>
<tr>
<td>source-destination pairs</td>
<td>10-9</td>
<td>10-9</td>
</tr>
<tr>
<td>avg. num. routing entries</td>
<td>53</td>
<td>104</td>
</tr>
<tr>
<td>99-th perc. num. routing entries</td>
<td>53</td>
<td>104</td>
</tr>
<tr>
<td>avg. routing stretch</td>
<td>1.036</td>
<td>1.051</td>
</tr>
<tr>
<td>99-th perc. routing stretch</td>
<td>1.333</td>
<td>1.158</td>
</tr>
</tbody>
</table>

5.6. CONCLUDING DISCUSSION

While the above results demonstrate that a technique representing in principle any point in the state-stretch trade-off spectrum can be used in the real world, they...
also show that the state-stretch trade-offs in some techniques are more suitable for sensornets than in other techniques.

To begin with, shortest-path routing ensures the smallest possible stretch, which means that packets routed using this technique exhibit the lowest end-to-end delays and highest delivery rates, and, globally, they consume the least amount of energy, bandwidth, and buffer space in the network. In addition, from the four evaluated techniques, shortest-path routing is the easiest one to implement. These are the reasons why shortest-path routing is the dominant intra-domain routing technique in the Internet. However, in contrast to Internet routers, sensor nodes are typically severely constrained in terms of memory and bandwidth. Consequently, the linear dependency of the routing state on the node population size precludes practical deployments of shortest-path routing in sensornets beyond some tens of nodes.

On the other end of the state-stretch spectrum, there is beacon vector routing. In theory, it offers a constant state, and thus, should be well-suited for resource-constrained sensor nodes. Yet, in practice, the state in beacon vector routing is large and it is not clear whether indeed it should not depend on the node population size. Such a dependency could be justified considering that with too little state, the stretch of beacon vector routing can be excessively large. Moreover, implementing beacon vector routing is relatively difficult and introduces some problems unforeseen by the authors of this technique. Therefore, while the original work on beacon vector routing [35] was the first to address the need for a point-to-point routing protocol for sensornets, compared to the other techniques, beacon vector routing seems rather unattractive.

The two techniques that seem most practical for sensornets are compact routing and hierarchical routing. Although they have the same objectives — significantly reducing routing state while increasing stretch only slightly — their accomplishment of these objectives differs. Hierarchical routing offers a substantially smaller state than compact routing. This means that it requires less memory to store the state and less bandwidth and energy to maintain it. Compact routing, in contrast, offers a slightly smaller stretch. This means that packets routed using this technique may exhibit slightly lower end-to-end delays and slightly higher delivery rates, and, globally, they may consume slightly less energy, bandwidth, and buffer space.

For these reasons, the choice of one of these two techniques depends on a particular application. Applications that route lots of packets would preferably use compact routing. Applications that require large networks, in turn, would likely employ hierarchical routing instead.

In this view, hierarchical routing may potentially be more promising for
sensornet applications. Its large reduction of state as compared to compact routing may likely provide even a few orders of magnitude better scalability, when analyzed just from the memory perspective. Many sensornet applications require high scalability to cover a sufficient number of sensing and actuation points. Moreover, a large reduction in state substantially decreases the bandwidth and energy necessary for maintaining that state. The slack bandwidth and energy can be used to improve other properties of a routing protocol, such as the latency of reacting to changes in the network. In this way, some performance metrics of the applications on top, like robustness or network lifetime, can be improved. On the other hand, as argued in Section 2.1, many sensornet applications require low data rates. This means that they do not need to route as many packets as, for instance, Internet routers. Therefore, a difference in stretch may not have a large effect on the performance of such applications, especially a small difference like between hierarchical routing and compact routing, which results from the geometric nature of sensornet deployments. All in all, considering that hierarchical routing is also not necessarily much more difficult to implement than compact routing, its merits can make it an ideal routing technique for many sensornet applications. Nevertheless, such a claim has yet to be verified in real-world application-oriented deployments.

Irrespective of whether hierarchical routing turns out to be more practical than compact routing, this chapter conveys two points. First, it further reinforces the thesis formulated in Section 1.4 for this dissertation: when compared with other point-to-point routing techniques, hierarchical routing is indeed one of the most compelling ones for sensornets. Therefore, the efforts to prove that thesis — devising, implementing, and evaluating appropriate hierarchical routing protocols — turned out not to be senseless in the end. Second, this chapter, and in particular, the routing library developed with Tahir Azim, suggests that implementations of different routing techniques do not need to differ drastically. Moreover, the techniques can be implemented in a way that allows for changing one for another, practically without changing the application code. This means that if two competing routing techniques are implemented according to the same standard, for example, they are compatible with the proposed Internet architecture for sensornets, it may be possible to switch between these techniques depending on application requirements and particular deployment settings.

I believe there are now enough theoretical and experimental results to justify developing standardized sensornet implementations of the most appealing small-state routing protocols, namely hierarchical routing and compact routing. Such implementations would allow for the next research step, that is, testing the protocols in actual sensornet applications. Our routing library can be a good
starting point for developing such standardized implementations. For this reason, we are considering making the library sources publicly available.
Chapter 6

Conclusion

Proving the thesis formulated in this dissertation, which is the goal of this concluding chapter, is best done by revisiting the results presented in the previous chapters. To this end, Section 6.1 presents a summary of the most important research contributions and emphasizes conclusions that are most relevant for the thesis. In addition, Section 6.2 discusses possible future work, including some open problems that may potentially become exciting avenues for future research.

To begin drawing conclusions, recall the research problem this dissertation aims to address. The goal of the dissertation is to develop a point-to-point routing protocol suitable for sensornets. Point-to-point routing has been recognized by prior and parallel research activities as important functionality for sensornets. However, due to the requirements of sensornet applications, the resulting severe resource constraints of sensor nodes, the way they are deployed, and the peculiarities of low-power wireless communication they employ, developing suitable protocols has turned out to be difficult in practice.
6.1. RESEARCH RESULTS

Apart from some other minor challenges, the major challenge the above research problem introduces is that a point-to-point routing protocol for sensornets has to simultaneously offer small routing state, small routing stretch (i.e., nearly optimal routing paths), and robustness, while also being self-managed to a large extent. These requirements are explained in more detail in Chapter 2, based on an analysis of existing and some envisioned sensornet applications. Moreover, the analysis constitutes a strong argument that they are unlikely to change with the technology progress in the near future. This is because, to be deployed more widely, sensornet hardware technology will continue emphasizing different aspects, such as low-power operation, form factor, and price per unit, leaving all the constraints and peculiarities of sensornets to be coped with in software. Consequently, there is every likelihood that the research results presented in this dissertation will remain relevant for extended periods of time.

An analysis of related work on point-to-point routing, as performed in the second part of Chapter 2, yields some crucial observations. On the one hand, there exists an entire spectrum of routing techniques that are potentially suitable for sensornets, promising robustness and self-management capabilities. On the other hand, theory on routing evidences that there is an inherent trade-off between routing state and routing stretch, which all such techniques have to make: in short, the smaller the state in a technique, the larger the stretch, and vice versa. Therefore, the spectrum of techniques can be divided into a few main regions depending on the particular state-stretch trade-off they involve. It can then be observed that one of the most promising regions in the techniques spectrum, the region corresponding to hierarchical routing, has not received much research attention prior to this dissertation. In particular, unlike other representative techniques of the state-stretch trade-off spectrum, to the best of my knowledge, no hierarchical routing protocol has been implemented and evaluated in sensornets.

It can be speculated that there are mainly two reasons for the scarcity of research on hierarchical routing for sensornets. First, because of ensuring small routing state, hierarchical routing can theoretically yield large stretch in some network topologies. Second, even disregarding the difficulties stemming from the resource constraints and the peculiarities of sensornets, hierarchical routing is quite difficult to implement, especially the mechanisms for maintaining a cluster hierarchy, which is a foundation of this routing technique. These two issues can make hierarchical routing potentially unappealing as a routing technique for sensornets, even despite its other merits.

This dissertation challenges this view and advocates hierarchical routing.
There are three main motivating factors behind hierarchical routing. First, hierarchical routing offers very small routing state. Second, while the stretch in hierarchical routing can theoretically be large if arbitrary network topologies are considered, due to embedding in physical space, sensornet topologies are typically “geometric,” which means that their diameter grows fast with growing node populations. Theory proved that in such topologies hierarchical routing can offer both small state and small stretch. Third, implementing hierarchical routing need not necessarily be more difficult than implementing other techniques. In particular, by relaxing some of the hierarchy properties, one can devise localized algorithms in which nodes self-organize into and autonomously maintain the hierarchy in a highly robust and efficient manner. Consequently, all in all, hierarchical routing has the potential to be a compelling point-to-point routing technique for sensornets.

To support the above argument, in Chapter 3 a practical hierarchical routing protocol for sensornets, called PL-Gossip, is introduced. To the best of my knowledge, PL-Gossip is the first hierarchical routing protocol that has been effectively implemented for sensornets. It decomposes hierarchical routing into three widely-recognized manageable abstractions corresponding to different aspects of routing in sensornets, namely link quality estimation, routing state maintenance, and packet forwarding, and explains how each of the abstractions can be implemented to support hierarchical routing.

The most challenging, albeit crucial routing abstraction in hierarchical routing is routing state maintenance, that is, the maintenance of the cluster hierarchy reflected in node routing tables and routing addresses. As mentioned above, the intricacy of this problem is likely one of the reasons that hierarchical routing has not been implemented and evaluated in sensornets prior to this dissertation. Therefore, PL-Gossip puts special emphasis on the cluster hierarchy maintenance problem.

To make the problem practically tractable, rather than aiming at optimal hierarchies, PL-Gossip focuses on best-effort ones. In particular, Chapter 3 illustrates how to customize the properties of a cluster hierarchy, such that it can be proved (cf. Appendix A) that the hierarchy (1) is suitable for hierarchical routing, (2) has the potential to offer small routing state, and (3) can be maintained by real sensor nodes. Such custom properties need not be always be the same as those in Chapter 3, though; in particular, they are changed in subsequent chapters of the dissertation, even to describe different hierarchy types. This is just one of the examples that, rather than being a fixed and monolithic protocol, PL-Gossip introduces ideas that are more broadly applicable.

To maintain the properties of a cluster hierarchy, the variant of PL-Gossip
presented in Chapter 3 proposes to use a combination of two simple concepts: local operations for modifying the hierarchy and asynchronous local gossiping for propagating such modifications to the affected nodes. This is sufficient to allow nodes running PL-Gossip to self-organize into and autonomously maintain a cluster hierarchy described by a custom set of properties. However, simple though these two concepts are, combining them to enable self-managed hierarchy maintenance poses a number of challenges. Such challenges are associated, for example, with consistently adopting hierarchy modifications by the affected nodes or with electing those lower-level clusters that will be promoted to higher-level clusters. A lot of effort is dedicated to provide solutions to these nontrivial challenges and to prove the correctness of the solutions analytically. Again, however, these solutions are simple, which overall facilitates implementation of PL-Gossip on real sensor nodes. All in all, Chapter 3 illustrates that one can indeed devise a practical self-managed hierarchical routing protocol for senornets.

The development of PL-Gossip enables evaluating hierarchical routing experimentally in various senornet settings, which is the subject of Chapter 4. The evaluation employs three experimental platforms: a custom high-level simulator, TOSSIM, a low-level simulator for sensor nodes with realistic models of low-power wireless communication, and an actual 50+ node testbed, which has been built at Vrije Universiteit Amsterdam (cf. Appendix B). Hierarchical routing is evaluated with respect to the aforementioned requirements for a senornet routing protocol: routing state, routing stretch, and robustness.

The evaluation shows that, even though it does not aim at optimal cluster hierarchies, PL-Gossip can indeed offer very small routing state. Moreover, despite such small state, in practice PL-Gossip can also offer small stretch, on average within approximately 50% of the optimal one, which is the consequence of the aforementioned geometric nature of senornets. Finally, PL-Gossip is also robust in that failures cause relatively little disruption and the protocol recovers from them relatively fast. These results suggest that hierarchical routing is indeed an appealing point-to-point routing technique for senornets.

Furthermore, the evaluation of hierarchical routing in Chapter 4 is not only limited to the presented PL-Gossip protocol. As argued above, rather than being a fixed and monolithic protocol, PL-Gossip constitutes a basis of a protocol and introduces ideas that are more broadly applicable and can be experimented with. Therefore, rather than only implementing PL-Gossip, an entire hierarchical routing library has been developed for the evaluation. The library defines a common framework for a hierarchical routing protocol for senornets, which draws from the lessons learned from PL-Gossip. At the same time, however,
it identifies a number of design points. By varying the solutions at these
design points, one can evaluate and compare different mechanisms proposed for
hierarchical routing, as well as some novel ideas. The source code of the library
has been made publicly available (cf. Appendix C.1).

The library enables conducting thorough evaluation of different design
decisions that affect routing state, routing stretch, and robustness of hierarchical
routing itself. One of such design points studied in Chapter 4 is the type and
the properties of a cluster hierarchy. In particular, it is shown that a landmark
hierarchy typically requires larger state than an area hierarchy, but in return offers
smaller stretch, on average within approximately 25% of the optimal one; this
can be further controlled by varying particular properties of the hierarchy. In
other words, by varying the type or the properties of a cluster hierarchy, one
can further explore the state-stretch trade-off within hierarchical routing itself.
Another design point studied in Chapter 4 is the method for propagating hierarchy
information. Apart from local asynchronous gossiping, adopted by the variant of
PL-Gossip from Chapter 3, a common flooding-based method and a novel hybrid
one are studied. In short, it is shown that different methods affect different aspects
of robustness, thus again, hierarchical routing can be customized to optimize a
particular metric of interest. All in all, not only does the evaluation of hierarchical
routing conducted in Chapter 4 confirm that hierarchical routing is appealing for
sensornets, but also shows that it can be customized for particular applications.

Since the research presented in Chapter 3 and Chapter 4 fills in the
aforementioned gap in the routing techniques spectrum, it makes it possible to
experimentally compare representative techniques from the entire spectrum. Such
an initial comparison is the subject of Chapter 5.

To perform the comparison, another routing library, this time covering various
techniques, has been implemented, as described in Chapter 5. The library involves
four such techniques, which together represent the entire state-stretch trade-off
spectrum: shortest-path routing, which requires the largest state but offers the
minimal stretch, compact and hierarchical routing, which at a different granularity
reduce state at the expense of stretch, and constant-state routing, which needs the
smallest state but delivers the largest stretch. The great majority of the code is
shared by all techniques, which makes the implementations of these techniques
uniform to a large extent. The library is evaluated in TOSSIM and on two
testbeds: a 50+-node testbed and a 100+-node testbed. The evaluation focuses
on the state-stretch trade-off.

Apart from some minor results, the evaluation makes two major contributions.
First, it illustrates a general property of sublinear-state routing techniques.
More specifically, due to the geometric nature of sensornet deployments, most
techniques with sublinear state offer a routing stretch that is close to the optimal one. In other words, in sensornets, a large reduction in routing state typically entails only a slight increase in routing stretch. This leads to the second contribution, which is directly related to hierarchical routing. More specifically, compared to the other representative techniques, hierarchical routing offers a large reduction in state with only a slight increase in stretch. In particular, compared to compact routing, hierarchical routing offers a state that allows for at least an order of magnitude better scalability, while compromising the stretch only slightly, by less than 10–15% on average, which for many sensornet applications will degrade performance insignificantly. All in all, apart from reinforcing the argument that hierarchical routing is an appealing technique for sensornets, Chapter 5 suggests that, from the representative techniques from state-stretch trade-off spectrum, hierarchical routing offers a trade-off that is arguably most appealing for many sensornet applications.

Let us thus conclude that the thesis formulated in this dissertation holds:

Hierarchical routing is a compelling point-to-point routing technique for large sensornets. In practice, not only does it offer small routing state, but also small routing stretch. Moreover, it is possible to provide robust, efficient, self-managed hierarchical routing protocols that work in the real world.

6.2. FUTURE WORK

In the face of the above results, the problem of providing a point-to-point routing protocol suitable for sensornets may seem largely solved. Due to embedding in physical space, the topologies formed by sensornets are “geometric.” In effect, in sensornets, routing techniques with sublinear state can offer a routing stretch that is close to the optimal one. It is thus sensible to substantially reduce routing state, because it allows for overcoming the resource constraints of sensor nodes, and hence, for deploying large networks, which are required by many envisioned sensornet applications. Providing small routing state is precisely the objective of hierarchical routing. Therefore, since it has been demonstrated that hierarchical routing can be implemented for sensor nodes and, compared to other techniques, performs well in practice, hierarchical routing is a compelling candidate for standardized routing solutions that could be adopted by many sensornet applications.

However, while one can use the implementations of protocols presented in this dissertation to develop such standardized routing solutions, this process
will likely generate new problems. The research presented in this dissertation, and for that matter, most research on point-to-point routing in sensornets, adopted research-oriented protocol implementations. Since the goal of such implementations is to facilitate experimentation, they often do not address issues that are not directly related to routing. However, when a protocol is integrated into a complete system, such issues have to be addressed. During this process, it may turn out that what has been considered an engineering issue is in fact an intricate research problem.

An example of such an issue may be maintaining consistent neighbor information with a growing network density. When considering scalability, most routing techniques are concerned with a growth in the node population accompanied by a growth in the network diameter. Therefore, most existing protocol implementations assume that node neighbor tables are large enough to hold representative neighbor information. Since the implementations are evaluated in isolation and not as a part of a system, during evaluation typically enough memory is allocated to store complete neighborhood information. It may turn out, however, that in the real world sensornets grow in terms of node density rather than diameter. In a home environment, for example, more and more appliances may gradually be equipped with sensornet technology, and thus, a node embedded in some appliance will have more and more neighbors. When a routing protocol works as a part of a system, allocating enough memory to store complete neighbor information may not be possible. In effect, a routing protocol may be forced to work with only partial neighborhood information. While it is not yet clear whether this will ever be a significant problem, if it turns out to be, choosing such neighborhood information that ensures correct operation of a routing protocol may warrant additional research.

Another similar example can be given for PL-Gossip. More specifically, it is not clear whether cluster dissolution heuristics will be necessary for PL-Gossip to work in real-world systems. To date, the only situation in which they might potentially have been necessary was provoked by a Byzantine failure caused by a bug in the compiler employed for sensor nodes. Such an event is thus arguably unlikely to occur in a production-quality system. However, integrating PL-Gossip into an actual system and testing it for extended periods of time may identify other situations in which cluster dissolution heuristics may be useful. Again, developing such heuristics may require in-depth research. There are a number of similar issues in PL-Gossip, hierarchical routing in general, and in other routing techniques. However, without standardized implementations of such techniques, which can be integrated and used with actual sensornet systems, identifying these issues may be difficult.
Consequently, developing a standardized implementation of some small-state routing technique, be it hierarchical routing, compact routing, or some other technique, would constitute a significant contribution. On the one hand, the proposed extended Internet architecture for sensornets [48, 46, 47] can greatly simplify protocol integration into complete sensornet-based systems. Several moderate-size real-world systems have been and are being developed based on this architecture. On the other hand, there is now enough information on the performance of different small-state routing techniques in sensornets to enable choosing potentially appealing techniques. Having an implementation of a small-state routing technique that adheres to the standards of the proposed Internet architecture could thus facilitate scaling existing systems using that architecture and building novel systems. The implementation process itself and subsequent system-level testing would in turn demonstrate how the selected technique performs in real-world applications, and would likely identify new significant research problems.

What the above paragraphs are thus advocating is switching from protocol- to systems-oriented research, at least regarding point-to-point routing for sensornets. On the research side, without being aware of the actual problems point-to-point routing introduces in real-world applications, we, as a community, will be trying to solve artificial ones. On the practical side, without appropriate routing technology for sensornets, building many envisioned systems will not be possible, and the adoption of sensornets in the real world will not be able to progress beyond some simple applications. These two vices have already started being attributed to the sensornet community [107, 125]. Consequently, with respect to point-to-point routing for sensornets, I believe that it is high time and there are now enough results to enable the transition from protocol- to systems-oriented research and development. Only such systems-oriented research will likely generate the new generation of important protocol-related problems.
Bibliography


[37] V. Fuller and T. Li, “Classless inter-domain routing (CIDR): The Internet address assignment and aggregation plan,” The Internet Society RFC 4632, August 2006.


[57] K. Iwanicki and M. van Steen, “Gossip-based self-management of a recursive area hierarchy for large wireless sensornets,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 4, pp. 562–576, April 2010. DOI: [http://dx.doi.org/10.1109/TPDS.2009.89](http://dx.doi.org/10.1109/TPDS.2009.89)

[58] X. Jiang, S. Dawson-Haggerty, P. Dutta, and D. Culler, “Design and implementation of


BIBLIOGRAPHY


Within this appendix, proofs of the lemmas proposed in this dissertation are assembled. Due to the recursive nature of hierarchical routing, most of the proofs employ mathematical induction. The proofs aim to be as formal as possible without becoming illegible. They all assume the common notation used throughout the dissertation. The proofs have been published as supplemental material of an earlier paper [57].

A.1. PROOF OF LEMMA 3.1

The Lemma: For each cluster and any two nodes, A and B, belonging to this cluster, there exists a path between A and B in the graph reflected by the neighbor relation that consists only of nodes belonging to the cluster.
PROOF: We will perform the proof by induction. Let us assume the common notation for a cluster: \( \mathcal{C}_V^i \) denotes a level-\( i \) cluster with head node \( V \).

**Basis:** \( i = 0 \). From Property [1] a level-0 cluster contains just one node: \( \mathcal{C}_A^0 = \{ A \} \). The path from \( A \) to \( B = A \) naturally exists, and moreover, it involves only \( A \in \mathcal{C}_A^0 \). Since \( \mathcal{C}_A^0 \) was chosen arbitrarily, the lemma holds for \( i = 0 \).

**Inductive step:** \( i = k + 1 \) (where \( k \geq 0 \)). Assume that the lemma holds for all levels \( \leq k \). Let us take an arbitrary level-\( i \) cluster, \( \mathcal{C}_V^i \), and two arbitrary nodes from this cluster, \( A, B \in \mathcal{C}_V^i \).

\[ \text{There are three possible situations:} \]

1): \( A \) and \( B \) belong to the same subcluster, \( \mathcal{C}_{U}^{i-1} \), of cluster \( \mathcal{C}_V^i \).

\[ \text{In this case, from the inductive assumption, there exists a path from } A \text{ to } B \text{ that involves only nodes from } \mathcal{C}_{U}^{i-1}. \text{ Because } \mathcal{C}_{U}^{i-1} \subseteq \mathcal{C}_V^i \text{ (Property [3]), the path consists only of nodes from } \mathcal{C}_V^i. \text{ Therefore, the lemma holds in this situation.} \]

2): \( A \) belongs to some subcluster \( \mathcal{C}_{U}^{i-1} \) of cluster \( \mathcal{C}_V^i \), while \( B \) belongs to the central subcluster of \( \mathcal{C}_V^i \), that is, \( \mathcal{C}_V^i \).
In this case, since the central subcluster of a cluster is adjacent to all other subclusters of the clusters, there exists node $A' \in \mathcal{C}_U^{i-1} \subseteq \mathcal{C}_V$ and node $B' \in \mathcal{C}_W^{i-1} \subseteq \mathcal{C}_V$ such that $A'$ and $B'$ are neighbors. Moreover, from the inductive assumption there exists a path from $A$ to $A'$ consisting only of nodes in $\mathcal{C}_U^{i-1}$, thus in $\mathcal{C}_V$, and likewise, a path from $B'$ to $B$ consisting only of nodes in $\mathcal{C}_W^{i-1}$, thus in $\mathcal{C}_V$. Therefore, by merging the path from $A$ to $A'$ with the edge from $A'$ to $B'$ and the path from $B'$ to $B$, one receives a path from $A$ to $B$ that consists only of nodes from $\mathcal{C}_V$. Consequently, the lemma holds in this situation. Similarly, the lemma holds when we change $A$ and $B$, as the graph reflected in the neighbor relation is undirected.

3): $A$ belongs to some subcluster $\mathcal{C}_U^{i-1}$ of cluster $\mathcal{C}_V$, while $B$ belongs to another subcluster $\mathcal{C}_W^{i-1}$ of cluster $\mathcal{C}_V$.

In this case, from the definition of the central subcluster, there exist nodes $A' \in \mathcal{C}_U^{i-1} \subseteq \mathcal{C}_V$, $B' \in \mathcal{C}_W^{i-1} \subseteq \mathcal{C}_V$, and $C', C'' \in \mathcal{C}_V^{i-1} \subseteq \mathcal{C}_V$ such that $A'$ and $C'$ are neighbors and $B'$ and $C''$ are neighbors. From the inductive assumption: first, there exists a path from $A$ to $A'$ that consists only of nodes from $\mathcal{C}_U^{i-1}$ thus from $\mathcal{C}_V$; second, there exists a path from $C'$ to $C''$ that consists only of nodes from $\mathcal{C}_V^{i-1}$.
thus from $C_i^V$; third, there exists a path from $B'$ to $B$ that consists only of nodes from $C_{i-1}^W$ thus from $C_i^V$. By merging these three paths with the edges between $A'$ and $C'$, and $C''$ and $B'$, one gets a path between $A$ and $B$ that consists only of nodes from $C_i^V$. In other words, the lemma holds also in this situation.

As these three situations exhaust all possible node configurations, there always exists a path from $A$ to $B$ that consists only of nodes belonging to $C_i^V$. Since $A$ and $B$ were chosen arbitrarily, such a path exists for any two nodes from $C_i^V$.

By applying mathematical induction to the basis and the inductive step, we prove that the lemma holds for all levels $i$. ■

A.2. PROOF OF LEMMA 3.2

The Lemma: A node from a level-$i$ cluster can reach some node in any adjacent level-$i$ cluster in at most $3^i$ hops in the graph reflected by the neighbor relation.

Proof: We will perform the proof by induction.

Basis: $i = 0$. Let us take two arbitrary adjacent level-0 clusters: $C_0^V$ and $C_0^U$. From Property 1, $C_0^V = \{V\}$ and $C_0^U = \{U\}$. $C_0^V$ and $C_0^U$ are adjacent, hence $V$ and $U$ are neighbors, that is, $V$ can reach $U$ in $1 = 3^0$ hop. Since $C_0^V$ and $C_0^U$ were chosen arbitrarily, the lemma is true for $i = 0$.

Inductive step: $i = k + 1$ (where $k \geq 0$). Assume that the lemma holds for all levels $\leq k$. Let us take two arbitrary adjacent level-$i$ clusters, $C_i^V$ and $C_i^U$, and an arbitrary node, $A \in C_i^V$. Let $B$ denote a node in $C_i^U$ that has a neighbor, $A'$, such that $A' \in C_i^V$ (the existence of $A'$ is guaranteed by the definition of adjacent clusters).
Consider level-$i$ subclusters, that is, $\mathcal{C}_X^{i-1}, \mathcal{C}_Y^{i-1} \subseteq \mathcal{C}_i$, and $\mathcal{C}_Z^{i-1} \subseteq \mathcal{C}_{i'}$, such that $A \in \mathcal{C}_X^{i-1}, A' \in \mathcal{C}_Y^{i-1}$, and $B \in \mathcal{C}_Z^{i-1}$.

There are the following three possible situations:

1): $X = Y$ (meaning that cluster $\mathcal{C}_X^{i-1}$ is adjacent to cluster $\mathcal{C}_Z^{i-1}$).

In this case, from the inductive assumption $A$ can reach some node (not necessarily $B$) from $\mathcal{C}_Z^{i-1} \subseteq \mathcal{C}_{i'}$ in at most $3^{i-1} < 3^i$ hops.

2): $\mathcal{C}_X^{i-1}$ is adjacent to $\mathcal{C}_Y^{i-1}$. 
In this case, from the inductive assumption $A$ can get to some node in $C_{i-1}^i$ in a most $3^{i-1}$ hops and any node from $C_{i-1}^i$ can get to a node from $C_{i-1}^i$ in at most $3^{i-1}$ hops. Consequently, $A$ can get to some node (not necessarily $B$) from $C_{i-1}^i \subseteq C_U^i$ in at most $2 \cdot 3^{i-1} < 3^i$ hops.

3): $C_{i-1}^X$ is not adjacent to $C_{i-1}^Y$, but, from Property $4$, $C_{i-1}^X$ and $C_{i-1}^Y$ are both adjacent to the central subcluster of $C_V^i$, that is, $C_{i-1}^Z$.

In this case, from the inductive assumption $A$ can get to some node in $C_{i-1}^i$ in at most $3^{i-1}$ hops. Likewise, any node from $C_{i-1}^i$ can get to a node from $C_{i-1}^i$ in at most $3^{i-1}$ hops and any node from $G_{i-1}^i$ can get to a node from $C_{i-1}^Z$ in at most $3^{i-1}$ hops. Therefore, $A$ can get to some node from $C_{i-1}^Z \subseteq C_U^i$ in at most $3 \cdot 3^{i-1} = 3^i$ hops.

As these three situations exhaust all possible node configurations, $A$ can always get to a node from $C_U^i$ in at most $3^i$ hops. Since $A$ was chosen arbitrarily, any node from $C_V^i$ can get to some node from $C_U^i$ in at most $3^i$. Because $C_V^i$ and $C_U^i$ were also chosen arbitrarily, the lemma is true for $i = k + 1$.

By applying mathematical induction to the basis and the inductive step, we prove the lemma for all $i$. Moreover, $3^i$ is a tight bound, that is, it is reachable for
some configurations (cf. situation 3).

\[ \square \]

A.3. PROOF OF LEMMA 3.3

**The Lemma:** The distance between the head nodes of two adjacent level-\(i\) clusters is at most \(3^i\) hops in the graph reflected by the neighbor relation.

**Proof:** Again, we will perform the proof by induction. Let \(d(A,B)\) denote the distance in hops between nodes \(A\) and \(B\) in the graph reflected by the neighbor relation.

**Basis:** \(i = 0\). Let us take two arbitrary adjacent level-0 clusters: \(C^0_V\) and \(C^0_U\).

From Property 1, \(C^0_V = \{V\}\) and \(C^0_U = \{U\}\). \(C^0_V\) and \(C^0_U\) are adjacent, thus \(V\) and \(U\) are neighbors, that is, \(d(V,U) = 1 = 3^0\). As \(C^0_V\) and \(C^0_U\) were chosen arbitrarily, the lemma is true for \(i = 0\).

**Inductive step:** \(i = k + 1\) (where \(k \geq 0\)). Assume that the lemma holds for all levels \(\leq k\). Let’s take two arbitrary adjacent level-\(i\) clusters: \(C^i_V\) and \(C^i_U\).

There are three possible situations:

1) The central subcluster of \(C^i_V\), that is, \(C^i_{V^{-1}}\), is adjacent to the central subcluster of \(C^i_U\), that is, \(C^i_{U^{-1}}\).
In this case, from the inductive assumption, \( d(V, U) \leq 3^{i-1} < 3^i \).

2): There exists a subcluster, \( C_{i-1}^X \subseteq C_i^V \), such that it is adjacent to both \( C_{i-1}^V \) and \( C_{i-1}^U \).

In this case, \( d(V, U) \leq d(V, X) + d(X, U) \). From the inductive assumption \( d(V, X), d(X, U) \leq 3^{i-1} \), thus \( d(V, U) \leq 2 \cdot 3^{i-1} < 3^i \). Moreover, the same is true for a symmetric situation, that is, \( C_{i-1}^X \subseteq C_i^U \), as the graph reflected in the neighbor relation is undirected.

3): There exist two subclusters, \( C_{i-1}^X \subseteq C_i^V \) and \( C_{i-1}^Y \subseteq C_i^U \), such that \( C_{i-1}^X \) is adjacent to \( C_{i-1}^V \). From Property 4, \( C_{i-1}^X \) is adjacent to \( C_{i-1}^V \), and \( C_{i-1}^Y \) is adjacent to \( C_{i-1}^U \).
In this case, \( d(V,U) \leq d(V,X) + d(X,Y) + d(Y,U) \). From the inductive assumption \( d(V,X), d(X,Y), d(Y,U) \leq 3^{i-1} \), thus \( d(V,U) \leq 3 \cdot 3^{i-1} = 3^i \).

As these three situations exhaust all possible node configurations, we always have \( d(V,U) \leq 3^i \). Because \( C^i_V \) and \( C^i_U \) were chosen arbitrarily, the lemma is true for \( i = k+1 \).

By applying mathematical induction to the basis and the inductive step, we prove the lemma for all \( i \). Moreover, \( 3^i \) is a tight bound, that is, it is reachable for some configurations (cf. situation 3).

\[ \square \]

**A.4. PROOF OF LEMMA 3.4**

**The Lemma:** The distance between any two members of a level-\( i \) cluster is at most \( 3^i - 1 \) hops in the graph reflected by the neighbor relation.

**Proof:** The proof is performed by induction as a simple extension of the proof presented in Section A.1 for Lemma 3.1. Let \( d(A,B) \) denote the distance in hops between nodes \( A \) and \( B \) in the graph reflected by the neighbor relation.

In the basis of proof A.1, as a level-0 cluster contains just one node, \( A \), \( d(A,A) = 0 = 3^0 - 1 \) hops. Therefore, Lemma 3.4 holds for \( i = 0 \).

In the inductive step of proof A.1, there are three situations. In situation 1, from the inductive assumption, \( d(A,B) \leq 3^{i-1} - 1 < 3^i - 1 \), thus Lemma 3.4 also holds. In situation 2, \( d(A,B) \leq d(A,A') + d(A',B') + d(B',B) \leq (3^{i-1} - 1) + 1 + (3^{i-1} - 1) = 2 \cdot 3^{i-1} - 1 < 3^i - 1 \). Therefore, Lemma 3.4 holds again. In situation 3, \( d(A,B) \leq d(A,A') + d(A',C') + d(C',C'') + d(C'',B') + d(B',B) \leq (3^{i-1} - 1) + 1 + (3^{i-1} - 1) + 1 + (3^{i-1} - 1) = 3 \cdot 3^{i-1} - 1 = 3^i - 1 \). Consequently, Lemma 3.4 holds as well.
Overall, by applying mathematical induction to the basis and the inductive step, we prove that the lemma holds for all levels \(i\). Moreover, \(3^i - 1\) is a tight bound (cf. situation 3).

\[\blacksquare\]

### A.5. PROOF OF LEMMA 3.5

**The Lemma:** For any nodes \(A\) and \(B\) and any moment in time, \(t\), if the correct labels of these nodes are equal at some level \(i\), that is, \(L^*_t(A)[i] = L^*_t(B)[i]\), then the lengths of these labels are equal, that is, \(\text{len}(L^*_t(A)) = \text{len}(L^*_t(B))\), and the labels are themselves equal at all levels higher than \(i\), that is, \(L^*_t(A)[j] = L^*_t(B)[j]\) for all \(i \leq j < \text{len}(L^*_t(A)) = \text{len}(L^*_t(B))\). In other words, when the correct labels are considered, Property 3 holds for all nodes.

Before proving the lemma formally, let us first develop an intuition that the lemma holds. To this end, assume that at some level \(i\) the correct labels of arbitrary nodes \(A\) and \(B\) have the identifier of node \(V\), that is, \(L^*_t(A)[i] = L^*_t(B)[i] = V\).

From the definition of a correct label, the \(i+1\)-st element of the correct label of node \(A\) is equal to the \(i+1\)-st element of the current label of node \(V\). Assume that this element is equal to the identifier of node \(U\). Hence, we have \(L^*_t(A)[i+1] = L_t(L^*_t(A)[i])[i+1] = L_t(V)[i+1] = U\).

From the same definition, however, the \(i+1\)-st element of the correct label of node \(B\) is equal to the \(i+1\)-st element of the current label of node \(V\), that is, to \(U\). Hence, we have \(L^*_t(B)[i+1] = L_t(L^*_t(B)[i])[i+1] = L_t(V)[i+1] = U = L^*_t(A)[i+1]\).
This means that the fact that the correct labels of \(A\) and \(B\) are equal at level \(i\) implies that these labels are equal at level \(i+1\). Applying the same reasoning iteratively, the labels are equal at level \(i+2\), \(i+3\), and so forth; in other words, the labels are equal at any level \(j \geq i\). In particular, if at some level \(i+k\), at which the correct labels of \(A\) and \(B\) are equal to the identifier of some node \(W\), the length of the current label of node \(W\) is \(i+k+1\), then the lengths of the correct labels of \(A\) and \(B\) are both equal to \(i+k+1\), that is, \(\text{len}(L^*_i(A)) = \text{len}(L^*_i(B)) = \text{len}(L_i(W)) = i+k+1\).

To sum up, intuitively, the lemma is true because if the correct labels of \(A\) and \(B\) are equal at some level, then the definition of a correct label involves using the current labels of the same nodes to obtain the correct labels of \(A\) and \(B\) at any higher level.

Another explanation is that the correct labels of nodes simply describe the freshest parent-child relationship between cluster heads at all levels. Since, from the Responsibility Rule, a level-\(i\) cluster head is the authoritative node for deciding which level-\(i+1\) cluster head is its parent, and hence the grandparent of its child level-\(i-1\) cluster heads, and the great-grandparent of its grandchild level-\(i-2\) cluster heads, and so on, all ((great-)grand) children of the cluster head will have the same grandparent, great-grandparent, and so on. In other words, their correct labels will be equal at all levels greater than \(i\).

While the above intuition is relatively straightforward, proving the lemma formally is more intricate. More specifically, again we will perform the proof...
using induction.

**Proof:** Let us take two arbitrary nodes, \( A \) and \( B \), and an arbitrary moment in time, \( t \). Let us also take an arbitrary level, \( i \), such that, at time \( t \), the correct label of \( A \) is equal at position \( i \) to the correct label of \( B \) at position \( i \), that is, \( L^*_t(A)[i] = L^*_t(B)[i] \). This, in particular, means that \( i < \min\left(\text{len}(L^*_t(A)), \text{len}(L^*_t(B))\right) \).

The inductive reasoning will be performed over a \( k \)-element suffix of the label of node \( A \), that is, over the \( k = \text{len}(L^*_t(A)) - i \) last elements of the label of \( A \).

**Basis:** \( k = 1 \), that is, \( i = \text{len}(L^*_t(A)) - 1 \). From the assumption of the lemma, the correct labels of nodes \( A \) and \( B \) are equal at level \( i \), that is, \( L^*_t(A)[i] = L^*_t(B)[i] = V \) for some node \( V \).

Therefore, for all the last \( k \) elements, that is, for all \( j \) such that \( \text{len}(L^*_t(A)) - k = i \leq j < \text{len}(L^*_t(A)) = i + 1 \), we have \( L^*_t(A)[j] = L^*_t(B)[j] \). To prove that the lemma holds, we thus just have to show that \( \text{len}(L^*_t(A)) = \text{len}(L^*_t(B)) \).

Let us consider the current label of node \( V \). From the definition of the correct label of node \( A \) and the fact that \( L^*_t(A)[i] = V \) and \( \text{len}(L^*_t(A)) = i + 1 \), we must have that the length of the current label of node \( V \) is \( \text{len}(L_t(V)) = i + 1 \).
Therefore, again from the definition of the correct label, this time of node $B$, and the fact that $L_t^*(B)[i] = V$ and $\text{len}(L_t^*(V)) = i + 1$ we must have that the length of the correct label of node $B$ is $\text{len}(L_t^*(B)) = i + 1$.

![Diagram of labels](image)

In other words, $\text{len}(L_t^*(A)) = \text{len}(L_t^*(B))$, which proves the basis.

**Inductive step:** $k > 1$, that is, $i = \text{len}(L_t^*(A)) - k$. From the assumption of the lemma, the correct labels of nodes $A$ and $B$ are equal at level $i$, that is, $L_t^*(A)[i] = L_t^*(B)[i] = V$ for some node $V$. Moreover, since $i = \text{len}(L_t^*(A)) - k$ and $k > 1$, then $\text{len}(L_t^*(A)) > i + 1$. This means that the correct label of node $A$ does not end at position $i + 1$, but has an identifier of some node, let us denote it $U$, at this position, that is, $L_t^*(A)[i + 1] = U$.

![Diagram of labels with U](image)

If we could prove that the correct label of node $B$ also has $U$ at position $i + 1$, that is, that $\text{len}(L_t^*(B)) > i + 1$ and $L_t^*(B)[i + 1] = U$, this would mean that we would need to prove the inductive step just for $k-1$-element label suffixes. This, however, is already true from the inductive assumption. Therefore, to prove the inductive step, we just need to show that $\text{len}(L_t^*(B)) > i + 1$ and $L_t^*(B)[i + 1] = U$.
How to prove that \( \text{len}(L^*_{t}(B)) > i + 1 \) and \( L^*_{t}(B)[i + 1] = U \)?

Let us consider the current label of node \( V \). From the definition of the correct label of node \( A \) and the fact that \( L^*_{t}(A)[i] = V \) and \( \text{len}(L^*_{t}(A)) > i + 1 \), we must have that the length of the current label of node \( V \) is \( \text{len}(L^*_{t}(V)) > i + 1 \). Moreover, from the same definition and the fact that \( L^*_{t}(A)[i] = V \) and \( L^*_{t}(A)[i + 1] = U \), we must have that \( V \) has the identifier of node \( U \) at position \( i + 1 \) of its current label, that is, \( L(V)[i + 1] = L^*_{t}(A)[i + 1] = U \).

Therefore, again from the definition of the correct label, this time of node \( B \), and the fact that \( L^*_{t}(B)[i] = V \) and \( \text{len}(L^*_{t}(V)) > i + 1 \) we must have that the length of the correct label of node \( B \) is \( \text{len}(L^*_{t}(B)) > i + 1 \). Moreover, from the same definition and the fact that \( L^*_{t}(B)[i] = V \) and \( L^*_{t}(B)[i + 1] = U \), we must have that \( B \) has the identifier of node \( U \) at position \( i + 1 \) of its correct label, that is, \( L^*_{t}(B)[i + 1] = L^*_{t}(B)[i + 1] = U \).

Therefore, we have \( \text{len}(L^*_{t}(A)), \text{len}(L^*_{t}(B)) > i + 1 \) and \( L^*_{t}(B)[i + 1] = L^*_{t}(B)[i + 1] \). In other words, not only are the labels of \( A \) and \( B \) equal at position \( i \), but also at position \( i + 1 \).

Therefore, since \( \text{len}(L^*_{t}(A)) - (i + 1) = k - 1 \), and since \( L^*_{t}(A)[i + 1] = L^*_{t}(B)[i + 1] \), we now just have to prove that the inductive step holds for \( k-1 \)-element label suffixes. This, however, is true from the inductive assumption, as
mentioned above. Consequently, by using this assumption, we get \( \text{len}(L_t^*(A)) = \text{len}(L_t^*(B)) \) and, for all \( j \) such that \( i \leq j < \text{len}(L_t^*(A)) \), we get \( L_t^*(A)[j] = L_t^*(B)[j] \), which proves the entire inductive step.

Overall, by applying mathematical induction to the basis and the inductive step, we prove that the lemma holds for all \( k \)-element label suffixes. Since the length of a label is finite, and since time \( t \), level \( i \), and nodes \( A \) and \( B \) were chosen arbitrarily, the lemma holds at any time, for any labels of an arbitrary length, and an for any level. In short, we have just proved the lemma.

\[ \blacksquare \]

### A.6. PROOF OF LEMMA 3.6

Proving this lemma is more demanding than proving the previous ones. We will thus do it in a number of steps.

As the first step, we will investigate what it means that the current label of a node has converged to the correct label the node should have. To this end, let us investigate the relationship between the current label, \( L_t(A) \), and update vector, \( U_t(A) \), of a node and the correct label, \( L_t^*(A) \), of the node, as formalized by the following supporting lemma.

**Lemma A.1** For any node \( A \), any time \( t \), and any level \( i \), if \( L_t(A)[i] = L_t^*(A)[i] = L_t(V)[i] = V \) and \( U_t(A)[i] = U_t(V)[i] \) for some node \( V \), then either \( \text{len}(L_t(A)) = \text{len}(L_t(V)) = \text{len}(L_t^*(A)) \) = \( i + 1 \) or \( \text{len}(L_t(A)) \), \( \text{len}(L_t(V)) \), \( \text{len}(L_t^*(A)) \) > \( i + 1 \) and \( L_t(A)[i + 1] = L_t(V)[i + 1] = L_t^*(A)[i + 1] \).

**PROOF:** Let us take an arbitrary node, \( A \), an arbitrary moment in time, \( t \), and an arbitrary level, \( i \), such that \( L_t(A)[i] = L_t^*(A)[i] = L_t(V)[i] = V \) and \( U_t(A)[i] = U_t(V)[i] \) for some node \( V \). In this situation, at level \( i \), the current label of node \( A \) has converged to the correct label the node should have at this level. Moreover, \( A \) also has the freshest information on its membership in the hierarchy at level \( i + 1 \) as implied by the equality of the two update vectors. We will show that these two facts simply mean that the current label of \( A \) has converged to the correct label \( A \) should have also at level \( i + 1 \), which proves the lemma.

To this end, let us focus on the set of equations related to the correct label of \( A \), that is, \( L_t^*(A)[i] = V \). From the definition of a correct label, we know that either:

- \( \text{len}(L_t(V)) = \text{len}(L_t^*(A)) = i + 1 \), or
- \( \text{len}(L_t(V)) > i + 1 \) and \( \text{len}(L_t(A)) > i + 1 \) and \( L_t(V)[i + 1] = L_t^*(A)[i + 1] = U \) for some \( U \).
Now, let us focus on the set of equations related to the current label and update vector of A, that is, \( L_i(A)[i] = L_i(V)[i] = V \) and \( U_i(A)[i] = U_i(V)[i] \). Let us consider the definition of an update vector, in particular, the part formalized by Invariant A.4 on page 249, we get that either:

- \( \text{len}(L_i(V)) = \text{len}(L_i(A)) = i + 1 \), or
- \( \text{len}(L_i(V)) > i + 1 \) and \( \text{len}(L_i(A)) > i + 1 \) and \( L_i(V)[i+1] = L_i(A)[i+1] = U \) for some \( U \).

As an explanation, the above logical statements stem from the fact that the \( i \)-th element of \( U_i(A) \) denotes the sequence number of the last known (by A) label update made at level \( i+1 \) by the head of the level-\( i \) cluster node A belongs to, that is, by node V. Therefore, since \( U_i(A)[i] = U_i(V)[i] \), then A knows about the very last update performed by V at level \( i+1 \), which means that their current labels are the same at level \( i+1 \).

As the resulting alternatives are similar, one can combine them thereby obtaining the following set of equations:

- \( \text{len}(L_i(A)) = \text{len}(L_i(V)) = \text{len}(L_i^*(A)) = i + 1 \), or
- \( \text{len}(L_i(A)), \text{len}(L_i(V)), \text{len}(L_i^*(A)) > i + 1 \) and \( L_i(A)[i+1] = L_i(V)[i+1] = L_i^*(A)[i+1] = U \) for some \( U \).

Since node A, time \( t \), and level \( i \) were chosen arbitrarily, the above result ends the proof.

Since we know the relationship between the current label and update vector of a node and the correct label of the node, we can obtain the sufficient condition for the current label to be considered as converged to the correct label.

**Lemma A.2** For any node A and any time \( t \), if for all levels \( 0 \leq j < \text{len}(L_i(A)) \) we have \( U_i(A)[j] = U_i(L_i(A)[j])[j] \), then \( L_i(A) = L_i^*(A) \), that is, the current label of A has converged to the correct label A should have.

**Proof:** To perform the proof, let us introduce the following definition. For any node A and any time \( t \), we will call the current label of A, that is, \( L_i(A) \), as \( k \)-converged if and only if for all \( 0 \leq j < \min(k, \text{len}(L_i(A))) \), we have \( U_i(A)[j] = U_i(L_i(A)[j])[j] \).

We will show by induction that if \( L_i(A) \) is \( k \)-converged, then it satisfies: for all \( 0 \leq j < \min(k+1, \text{len}(L_i(A))) \), \( L_i(A)[j] = L_i^*(A)[j] \). This will prove the
lemma because, in particular, if \( k \geq \text{len}(L_t(A)) \), then a \( k \)-converged label is simply converged, that is, \( L_t(A) = L^*_t(A) \).

To this end, let us take an arbitrary node, \( A \), and arbitrary time, \( t \).

**Basis:** \( k = 1 \). From the definition of a \( k \)-converged label, we have \( U_t(A)[0] = U_t(L_t(A)[0])[0] \). From Invariant \([A.1] \) we know that \( L_t(A)[0] = A \). From the definition of the correct label, we know that \( L^*_t(A)[0] = A \). To sum up, we know that, \( L_t(A)[0] = L^*_t(A)[0] = A \).

Therefore, from Lemma \([A.1] \) we know that either:

- \( \text{len}(L_t(A)) = \text{len}(L^*_t(A)) = 1 \), or
- \( \text{len}(L_t(A)), \text{len}(L^*_t(A)) > 1 \) and \( L_t(A)[1] = L^*_t(A)[1] \).

This, however, means that for all \( 0 \leq j < \min \left(1 + 1, \text{len}(L_t(A))\right) \), \( L_t(A)[j] = L^*_t(A)[j] \). In other words, the lemma holds for the basis.

**Inductive step:** \( k > 1 \). From the definition of a \( k \)-converged label: for all \( 0 \leq j < \min \left(k, \text{len}(L_t(A))\right) \), we have \( U_t(A)[j] = U_t(L_t(A)[j])[j] \) (let us denote this \( \dagger \)). Therefore, from the inductive assumption: for all \( 0 \leq j < \min \left(k - 1 + 1, \text{len}(L_t(A))\right) \), \( L_t(A)[j] = L^*_t(A)[j] \) (let us denote this \( \ddagger \)).

If \( k \geq \text{len}(L_t(A)) \), then the lemma holds because for all \( 0 \leq j < \min \left(k, \text{len}(L_t(A))\right) \) we have \( L_t(A)[j] = L^*_t(A)[j] \) (\( \ddagger \)) and \( U_t(A)[j] = U_t(L_t(A)[j])[j] \) (\( \dagger \)), in particular, we have \( L_t(A)[k - 1] = L^*_t(A)[k - 1] = V \) and \( U_t(A)[k - 1] = U_t(L_t(A)[k - 1])[k - 1] = U_t(V)[k - 1] \) for some \( V \). Consequently, from Lemma \([A.1] \) we know that either:

- \( \text{len}(L_t(A)) = \text{len}(L_t(V)) = \text{len}(L^*_t(A)) = k \), or
- \( \text{len}(L_t(A)), \text{len}(L_t(V)), \text{len}(L^*_t(A)) > k \) and \( L_t(A)[k] = L_t(V)[k] = L^*_t(A)[k] = U \) for some \( U \).

Since \( k < \text{len}(L_t(A)) \), we consider only the second part of the above alternative: \( \text{len}(L_t(A)), \text{len}(L^*_t(A)) > k \) and \( L_t(A)[k] = L^*_t(A)[k] \). Putting this together with \( \dagger \) and \( \ddagger \) we know that: for all \( 0 \leq j < \min \left(k + 1, \text{len}(L_t(A))\right) \), \( L_t(A)[j] = L^*_t(A)[j] \). In other words, the lemma holds for the inductive step.
Overall, by applying mathematical induction to the basis and the inductive step, we prove that the lemma holds for all \( k \). Since node \( A \) and time \( t \) were chosen arbitrarily, the lemma holds in general.

Essentially, Lemma A.2 illustrates a perfect match between the correct label of a node and a pair: the current label and update vector, which has been the intention of the update propagation mechanisms introduced for PL-Gossip. Oversimplifying things, if a node knows about the freshest label updates made by the heads of its clusters at all levels, then its current label is equal to the correct label.

We know now when the current label of a node has converged to the correct label the node should have. Yet, we do not know how the correct label looks like. In particular, in a normal system, the correct label of a node can change virtually anytime due to changes in the cluster hierarchy. Proving anything in such a dynamic network would be difficult.

Therefore, to prove the correctness of the of the update propagation mechanisms, in the main lemma we assumed that the network becomes quiescent at time \( t_q \), that is, starting from \( t_q \), there are no changes in the connectivity graph reflected in the neighbor relation and no label extension or label cut operations are executed by any node. We will show that in such a network, the correct labels of the nodes do not change. We will also demonstrate, how such correct labels look like. This is formalized by the following supporting lemma.

**Lemma A.3** In a network that becomes quiescent at time \( t_q \), the correct label of a node does not change, that is, for any node \( A \) and any time \( t \geq t_q \), \( L^*_t(A) = L^*_{t_q}(A) \).

**Proof:** Let us assume that the network becomes quiescent at time \( t_q \). We will prove the lemma by contradiction.

To this end, suppose that at some time \( t_{ch} \geq t_q \), the correct label of some node \( A \) changes. Let \( i_{ch} \) denote the minimal level of such a change.

Let us first consider the case in which \( i_{ch} = 0 \). From the definition of the correct label, for any time \( t \), \( \text{len}(L^*_t(A)) > 0 \) and \( L^*_t(A)[0] = A \). Hence, in particular, \( \text{len}(L^*_t(A)) > 0 \) and \( L^*_t(A)[0] = A \).

The condition \( i_{ch} = 0 \) would mean that either \( \text{len}(L^*_{t_{ch}}(A)) = 0 \), or \( \text{len}(L^*_{t_{ch}}(A)) > 0 \) but \( L^*_{t_{ch}}(A)[0] \neq A \). Any of these two, however, would mean that the correct label of node \( A \) at time \( t_{ch} \) does not satisfy the definition of the correct label — **contradiction**. Therefore, we are allowed to consider only the case in which \( i_{ch} > 0 \).

Let us thus assume that \( i_{ch} > 0 \) and \( L^*_t(A)[i_{ch} - 1] = V \) for some node \( V \). From the fact that \( i_{ch} \) is the minimal level at which the change occurred, we have
Let us thus focus on the current label of node $V$. From the definition of the correct label of node $A$, node $V$ is the head of a level $i_{ch}-1$ cluster, $C_{i_{ch}-1}$, that is, $L_q(V)[i_{ch}-1] = L_{i_{ch}}(V)[i_{ch}-1] = V$, and also the current label of node $V$ has to satisfy either of the conditions below:

- $\text{len}(L_q(V)) = i_{ch}$ (and then $\text{len}(L^*_q(A)) = i_{ch}$), or
- $\text{len}(L_q(V)) > i_{ch}$ and $L_q(V)[i_{ch}] = U$ for some $U$ (and then $\text{len}(L^*_q(A)) > i_{ch}$ and $L^*_q(A)[i_{ch}] = U$).

The fact that, at time $t_{ch}$, $L^*_q(A)$ differs from $L^*_q(A)$ at level $i_{ch}$ and that $i_{ch}$ is the minimal such level implies either of the following situations for all $t_q \leq t < t_{ch}$:

1. $\text{len}(L_q(V)) > i_{ch}$ and $\text{len}(L_{i_{ch}}(V)) > i_{ch}$ and $L_q(V)[i_{ch}] \neq L_{i_{ch}}(V)[i_{ch}]$, or
2. $\text{len}(L_q(V)) = i_{ch}$ and $\text{len}(L_{i_{ch}}(V)) > i_{ch}$, or
3. $\text{len}(L_q(V)) > i_{ch}$ and $\text{len}(L_{i_{ch}}(V)) = i_{ch}$.

In a quiescent system, the only operation executed on node labels is the label combination operation. Therefore, only label combination executed by node $V$ upon reception of a heartbeat message from some node $W$ could cause any of situations 1–3 at time $t_{ch}$. However, using Hoare logic on the pseudo-code of the label combination operation from Listing 3.6 on page 229, we will show that it is impossible, which means that neither of the three situations is possible.

To this end, let us assume that $L_{t_{ch}}(V)$ and $U_{t_{ch}}(V)$ denote, respectively, the current label and update vector of node $V$ after the label combination operation that caused the change at level $i_{ch}$. Conversely, let us assume that $L_{t_{ch}-\epsilon}(V)$ and $U_{t_{ch}-\epsilon}(V)$ denote, respectively, the current label and update vector of node $V$ immediately before the label combination operation that caused the change at level $i_{ch}$. Assume also that $L_{t_{ch}}(W)$, $U_{t_{ch}}(W)$, $L_{t_{ch}-\epsilon}(W)$, and $U_{t_{ch}-\epsilon}(W)$ represent the corresponding data for node $W$ that broadcast the heartbeat.

Let us now rewrite the label combination listing from page 229 with this new notation. The proof that the new listing is equivalent to the old one is trivial, albeit laborious, so it is left to the reader.

```plaintext
... {L_{t_{ch}}(V) = L_{t_{ch}-\epsilon}(V) and U_{t_{ch}}(V) = U_{t_{ch}-\epsilon}(V)}
{L_{t_{ch}}(W) = L_{t_{ch}-\epsilon}(W) and U_{t_{ch}}(W) = U_{t_{ch}-\epsilon}(W)}
8 i ← 0;
9 copying ← false;
10 while i < min(\text{len}(L_{t_{ch}-\epsilon}(V)), \text{len}(L_{t_{ch}-\epsilon}(W))) do begin
```
if copying then begin
  \text{headID} \text{ variable is no longer necessary }
end;
if \text{LTch}(V)[i] = \text{LTch}(W)[i] then begin
  if \text{LTch}(V)[i] < \text{LTch}(W)[i] then copying := true
else copying := false;
end;

\text{LTch}(V)[i-1] \leftarrow \text{LTch}(W)[i-1];
\text{Uttch}(V)[i-1] \leftarrow \text{Uttch}(W)[i-1];

end;

\text{LTch}(V)[i] \leftarrow \text{LTch}(W)[i];
\text{Uttch}(V)[i] \leftarrow \text{Uttch}(W)[i];
\text{LTch}(V)[i+1] \leftarrow \text{LTch}(W)[i+1];
\text{Uttch}(V)[i+1] \leftarrow \text{Uttch}(W)[i+1];

\text{while } \text{LTch}(V)[i] < \text{LTch}(W)[i] do begin
  \text{LTch}(V)[i] \leftarrow \text{LTch}(W)[i];
  \text{Uttch}(V)[i] \leftarrow \text{Uttch}(W)[i];
  i \leftarrow i + 1;
end;

\{ \text{LTch}(W) = \text{LTch}(V) \text{ and Uttch}(W) = \text{Uttch}(V) \}

\ldots

\ldots

\ldots

\text{Situation 1.:} \text{len}(L_{\text{th}}(V)) > i_{\text{ch}} \text{ and len}(L_{\text{th}}(V)) > i_{\text{ch}} \text{ and } L_{\text{th}}(V)[i_{\text{ch}}] \neq L_{\text{th}}(V)[i_{\text{ch}}]. \text{ The only places in the above listing at which } L_{\text{th}}(V)[i] \text{ is assigned are lines 13 and 27:}

\ldots

13 \qquad L_{\text{th}}(V)[i] \leftarrow L_{\text{th}}(W)[i];

\ldots

27 \qquad L_{\text{th}}(V)[i] \leftarrow L_{\text{th}}(W)[i];

\ldots

The assignment can happen only if the \textit{copying} variable is set to \textit{true}.

\ldots

12 \quad \text{if copying then begin}

\{ \text{copying = true} \}

13 \quad L_{\text{th}}(V)[i] \leftarrow L_{\text{th}}(W)[i];

\ldots

15 \quad \text{end};

\ldots

22 \quad \text{if copying then begin}

\{ \text{copying = true} \}

27 \quad L_{\text{th}}(V)[i] \leftarrow L_{\text{th}}(W)[i];

\ldots

\ldots
The \textit{copying} variable is modified only in lines 9, 17, and 18:
\begin{verbatim}
8     i ← 0;
9     copying ← false;
10    while i < \min\left(\text{len}(L_{t,h-\varepsilon}(V)),\text{len}(L_{t,h-\varepsilon}(W))\right) do begin
\end{verbatim}
\begin{verbatim}
16       \textbf{if} \quad \text{LT}_{t,h-\varepsilon}(V)[i] = \text{LT}_{t,h-\varepsilon}(W)[i] \quad then begin
17         \textbf{if} \quad \text{UT}_{t,h-\varepsilon}(V)[i] < \text{UT}_{t,h-\varepsilon}(W)[i] \quad then copying ← true
18      \textbf{else} \quad copying ← false;
19      \textbf{end};
20     i ← i + 1;
21    end;
\end{verbatim}
\begin{verbatim}
Based on these cases, we can formulate the following Hoare invariant for the \textit{copying} variable.
\begin{equation}
(\dagger): \text{copying} = \text{true} \quad \text{if and only if}
\begin{align*}
\text{there exists} \quad & i_{\text{cop}} \quad \text{such that} \quad (0 \leq i_{\text{cop}} < i \quad \text{and} \\
& \text{LT}_{t,h-\varepsilon}(V)[i_{\text{cop}}] = \text{LT}_{t,h-\varepsilon}(W)[i_{\text{cop}}] \quad \text{and} \\
& \text{UT}_{t,h-\varepsilon}(V)[i_{\text{cop}}] < \text{UT}_{t,h-\varepsilon}(W)[i_{\text{cop}}] \quad \text{and} \\
& \text{for all} \quad i_{\text{cop}} < j < i \quad (\text{LT}_{t,h-\varepsilon}(V)[j] \neq \text{LT}_{t,h-\varepsilon}(W)[j])
\end{align*}
\end{equation}
Essentially, the invariant says that the \textit{copying} variable can be set to true if and only if at some level \(i_{\text{cop}}\) node \(V\) has older information than node \(W\) about the membership of the common level-\(i_{\text{cop}}\) cluster in the hierarchy. The invariant holds at the beginning of the main \textbf{while} loop, between lines 10 and 11, and after the loop, between lines 21 and 22:
\begin{verbatim}
10    \textbf{while} \quad i < \min\left(\text{len}(L_{t,h-\varepsilon}(V)),\text{len}(L_{t,h-\varepsilon}(W))\right) \quad \textbf{do begin}
\end{verbatim}
\begin{verbatim}
11      \{ \text{(\dagger) holds} \}
12     \textbf{if} \quad \text{copying} \quad \textbf{then begin}
\end{verbatim}
\begin{verbatim}
21     \textbf{end};
\end{verbatim}
\begin{verbatim}
22      \{ \text{(\dagger) holds as well} \}
\end{verbatim}
Proving this fact using Hoare logic is trivial, albeit laborious. For this reason, such a proof is left to the reader.
From the above analysis, we know that Situation 1. can take place as a result of the above label combination operation, that is, \( \text{len}(L_{t_{ch}}(V)) > i_{ch} \) and \( \text{len}(L_{t_{ch}}(V)) > i_{ch} \) and \( L_{t_{ch}}(V)[i_{ch}] \neq L_{t_{ch}}(V)[i_{ch}] \), if and only if there exists some \( i_{cop} \), such that \( 0 \leq i_{cop} < i_{ch} \) and \( L_{t_{ch}}(V)[i_{cop}] = L_{t_{ch}}(V)[i_{cop}] \) and \( U_{t_{ch}}(V)[i_{cop}] < U_{t_{ch}}(W)[i_{cop}] \) and, for all \( i_{cop} < i \), \( L_{t_{ch}}(V)[j] \neq L_{t_{ch}}(W)[j] \).

However, from the fact that \( L_{t_{ch}}(V)[i_{ch} - 1] = L_{t_{ch}}(V)[i_{ch} - 1] = V \) and Invariant \( \text{A.2} \) on page 249, we know that for all \( 0 \leq j < i_{ch} \), \( L_{t_{ch}}(V)[j] = L_{t_{ch}}(V)[j] = V \). Combined with Invariant \( \text{A.3} \) on page 249, this yields that for all \( 0 \leq j < i_{ch} \), for any active node \( V' \), if \( L_{t_{ch}}(V')[i] = L_{t_{ch}}(V)[i] = V \), then \( U_{t_{ch}}(V)[i] \geq U_{t_{ch}}(V')[i] \).

Therefore, even if there exists \( i_{cop} \), such that \( 0 \leq i_{cop} < i_{ch} \) and \( L_{t_{ch}}(V)[i_{cop}] = L_{t_{ch}}(W)[i_{cop}] \), we cannot have \( U_{t_{ch}}(V)[i_{cop}] < U_{t_{ch}}(W)[i_{cop}] \); instead, we will always have \( U_{t_{ch}}(V)[i_{cop}] \geq U_{t_{ch}}(W)[i_{cop}] \), because the head of a cluster always has the freshest information on the membership of the cluster in the hierarchy. This means that Situation 1. is impossible, so let us progress to the next two situations.

**Situation 2. and 3.:** \( \text{len}(L_{t}(V)) = i_{ch} \) and \( \text{len}(L_{t_{ch}}(V)) > i_{ch} \), or \( \text{len}(L_{t}(V)) > i_{ch} \) and \( \text{len}(L_{t_{ch}}(V)) = i_{ch} \). The only place in the the label combination listing at which \( \text{len}(L_{t_{ch}}(V)) \) is assigned is line 24:

\[
\ldots
24 \quad \text{len}(L_{t_{ch}}(V)) \leftarrow \text{len}(L_{t_{ch}}(W)) ;
\]

Again, this can happen only if the \textit{copying} variable is set to \textit{true}:

\[
\ldots
22 \quad \text{if} \; \text{copying} \; \text{then} \; \text{begin} \\
\ldots
\quad \{ \text{copying} = \text{true} \} \\
24 \quad \text{len}(L_{t_{ch}}(V)) \leftarrow \text{len}(L_{t_{ch}}(W)) ;
\]

\[
\ldots
31 \quad \text{end} ;
\]

Again, the (\( \sharp \)) invariant holds for the \textit{copying} variable. Therefore, again, neither Situation 2. nor Situation 3. are possible.

To sum up, neither of the three situations are possible — \textbf{CONTRADICTION}. Therefore, in a quiescent network, there is no \( i_{ch} \) at which the current label of node \( V \) can change at time \( t_{ch} \geq t_{q} \) as a result of receiving a heartbeat message from some node \( W \). Since nodes \( V \) and \( W \) were chosen arbitrarily for node \( A \), there is no \( i_{ch} \) at which the correct label of node \( A \) can change at time \( t_{ch} \geq t_{q} \) as a result of some label combination operation anywhere in the network. Since
the label combination operation is the only operation using which node labels can
be modified in a quiescent network, the correct label of node $A$ cannot change at
time $t_{ch} \geq t_q$ in a network that is quiescent at time $t_q$. Since node $A$ and time $t_{ch}$
were chosen arbitrarily, the supporting lemma holds. ■

We know that in a quiescent system the correct label of a node does not change
and is equal to the correct label of the node at time $t_q$ when the network became
quiescent (Lemma A.3). We also know that the current label of the node has
converged to the correct label if the current update vector contains the freshest
updates at all levels (Lemma A.2). We will demonstrate now that, once (a part
of) the current label has converged to the correct label, it will never drift from
the correct label in a quiescent system, as formalized by the following supporting
lemma.

Lemma A.4 In a network that becomes quiescent at time $t_q$, for any node $A$, any
time $t \geq t_q$, any time $t' \geq t$, and any level $k \leq \text{len}(L_t(A))$, if for all $0 \leq j < k$ we
have $U_t(A)[j] = U_t(L_t(A)[j])[j]$, then $\text{len}(L_{t'}(A)) \geq k$ and for all $0 \leq j < k$ we
have $L_{t'}(A)[j] = L_t(A)[j]$ and $U_{t'}(A)[j] = U_t(A)[j]$.

Proof: Let us assume that the network becomes quiescent at time $t_q$. Like the
previous one, we will perform this proof by contradiction using Hoare logic on
the listing of the label combination operation.

To this end, suppose that the lemma assumptions holds, that is, for some node
$A$, at some time $t \geq t_q$, and for some level $k \leq \text{len}(L_t(A))$, for all $0 \leq j < k$ we
have $U_t(A)[j] = U_t(L_t(A)[j])[j]$.

However, assume also that despite the lemma assumptions are satisfied, the
lemma itself does not hold, that is, at some time $t' \geq t$ we have $\text{len}(L_{t'}(A)) < k$ or
for some $0 \leq j_{ch} < k$ we have $L_{t'}(A)[j_{ch}] \neq L_t(A)[j_{ch}]$ or $U_{t'}(A)[j_{ch}] \neq U_t(A)[j_{ch}]$.

Finally, assume also that $t'$ is the minimal time after $t$ at which the lemma
itself does not hold; in other words, for any earlier time $t \leq t'' < t'$, the lemma
holds, that is, $\text{len}(L_{t''}(A)) \geq k$ and for all $0 \leq j < k$ we have $L_{t''}(A)[j] = L_t(A)[j]$ and
$U_{t''}(A)[j] = U_t(A)[j]$.

In a quiescent system, the only operation executed on node labels is the label
combination operation. Therefore, only label combination executed by node $A$
upon reception of a heartbeat message from some node $B$ could potentially cause
the lemma not to hold at time $t'$. However, using Hoare logic on the pseudo-code
of the label combination operation from Listing 3.6 on page 234, we will show that
such a situation is impossible, which means that the lemma must hold.

To this end, let us assume that $L_{t'}(A)$ and $U_{t'}(A)$ denote, respectively,
the current label and update vector of node $A$ after the label combination
operation that caused one of the situations. Conversely, let us assume that \( L'_{\ell - \epsilon}(A) \) and \( U'_{\ell - \epsilon}(A) \) denote, respectively, the current label and update vector of node \( A \) immediately before the label combination operation that invalidated the lemma. Assume also that \( L'_{\ell}(B) \), \( U'_{\ell}(B) \), \( L'_{\ell - \epsilon}(B) \), and \( U'_{\ell - \epsilon}(B) \) represent the corresponding data for node \( B \) that broadcast the heartbeat.

Like in the previous proof, let us now rewrite the label combination listing from page 94 with this new notation. Again, the proof that the new listing is equivalent to the old one is trivial, albeit laborious, so it is left to the reader.

\[
\{L'_{\ell}(A) = L'_{\ell - \epsilon}(A) \text{ and } U'_{\ell}(A) = U'_{\ell - \epsilon}(A)\}\\
\{L'_{\ell}(B) = L'_{\ell - \epsilon}(B) \text{ and } U'_{\ell}(B) = U'_{\ell - \epsilon}(B)\}
\]

Note that the (\( \frac{3}{4} \)) invariant from the previous proof holds also here for the main \( \text{while} \) loop, that is, it holds at the beginning of the loop and after the loop. We will modify the invariant slightly to make it hold in the whole listing, starting
Again, proving that the generalized invariant holds is trivial, but laborious. Therefore, the proof is left to the reader.

From the lemma assumption, we know that at time $t' - \varepsilon$, $\text{len}(L_{t' - \varepsilon}(A)) \geq k$ and, for all $0 \leq j < k$, $U_{t' - \varepsilon}(A)[j] = U_{t' - \varepsilon}(L_{t' - \varepsilon}(A))$. Therefore, from Invariant A.3 on page 249, we know that, for all $0 \leq j < k$, $U_{t' - \varepsilon}(A)[j] \geq U_{t' - \varepsilon}(B)[j]$.

We can apply this knowledge to the generalized (§) invariant, obtaining a new version of the invariant, denoted (§), that holds in the whole listing, starting from line 10:

(§) : \text{copying} = \text{true} \quad \text{if and only if}

\begin{align*}
\text{there exists } & i_{\text{cop}} \text{ such that } \\
0 & \leq i_{\text{cop}} < \min(i, \text{len}(L_{t' - \varepsilon}(A)), \text{len}(L_{t' - \varepsilon}(B))) \quad \text{and} \\
\text{len}(L_{t' - \varepsilon}(A)[i_{\text{cop}}]) & = \text{len}(L_{t' - \varepsilon}(B)[i_{\text{cop}}]) \quad \text{and} \\
U_{t' - \varepsilon}(A)[i_{\text{cop}}] & < U_{t' - \varepsilon}(B)[i_{\text{cop}}] \quad \text{and} \\
\text{for all } & i_{\text{cop}} < j < \min(i, \text{len}(L_{t' - \varepsilon}(A)), \text{len}(L_{t' - \varepsilon}(B))) \\
(L_{t' - \varepsilon}(A)[j] & \neq L_{t' - \varepsilon}(B)[j])
\end{align*}

We know that after the label combination operation, the lemma no longer holds, that is, $\text{len}(L_t(A)) < k$, or for some $0 \leq j_{ch} < k$ we have $L_t(A)[j_{ch}] \neq L_t(A)[j_{ch}]$ or $U_t(A)[j_{ch}] \neq U_t(A)[j_{ch}]$.

Let us thus focus on the first case: $\text{len}(L_t(A)) < k$. Since the lemma holds before the label combination operation, we know that $\text{len}(L_{t' - \varepsilon}(A)) \geq k$. The only operation in the listing that can modify the length of the label of $A$ is the assignment in line 24. Since after the assignment $\text{len}(L_t(A)) = \text{len}(L_{t' - \varepsilon}(B))$ and since $\text{len}(L_t(A)) < k$, we must have $\text{len}(L_{t' - \varepsilon}(B)) < k$.

Let us thus look at the preconditions of the assignment in line 24, which we obtain using the rules of Hoare logic. First, the (§) invariant must hold, as it holds
in the whole listing starting from line 10. Second, the copying variable must be set to true; otherwise, the \( i \) branch in line 22 that contains the assignment is not entered. Third, from the negation of the main while loop condition in line 10, \( i \geq \min \left( \text{len}(L'_{i-\varepsilon}(A)), \text{len}(L'_{i-\varepsilon}(B)) \right) \); otherwise, the while loop cannot have finished.

If we combine these three preconditions with the fact that \( \text{len}(L'_{i-\varepsilon}(B)) < k \), we will realize that if the assignment takes place, there must exist \( i_{\text{cop}} \) such that:

\[
\begin{align*}
\text{len}(L'_{i-\varepsilon}(B)) & \leq i_{\text{cop}} \quad \text{ and } \\
i_{\text{cop}} & < \text{len}(L'_{i-\varepsilon}(B)) \quad \text{ and } \\
L'_{i-\varepsilon}(A)[i_{\text{cop}}] & = L'_{i-\varepsilon}(B)[i_{\text{cop}}] \quad \text{ and } \\
U'_{i-\varepsilon}(A)[i_{\text{cop}}] & < U'_{i-\varepsilon}(B)[i_{\text{cop}}] \quad \text{ and } \\
\text{for all } i_{\text{cop}} < j < \min \left( \text{len}(L'_{i-\varepsilon}(A)), \text{len}(L'_{i-\varepsilon}(B)) \right) & \left( L'_{i-\varepsilon}(A)[j] \neq L'_{i-\varepsilon}(B)[j] \right)
\end{align*}
\]

This means that, in particular, \( i_{\text{cop}} \) must simultaneously satisfy \( \text{len}(L'_{i-\varepsilon}(B)) \leq i_{\text{cop}} \) and \( \text{len}(L'_{i-\varepsilon}(B)) > i_{\text{cop}} \). Such an \( i_{\text{cop}} \) does not exist. Therefore, the assignment cannot invalidate the lemma — **CONTRADICTION.** This means that the first part of the lemma will always hold after the label combine operation, that is, \( \text{len}(L_{i}(A)) \geq k \).

Let us thus focus on the second case in which the lemma might potentially not hold: for some \( 0 \leq j_{\text{ch}} < k \) we have \( L'_{i}(A)[j_{\text{ch}}] \neq L_{i}(A)[j_{\text{ch}}] \) or \( U'_{i}(A)[j_{\text{ch}}] \neq U_{i}(A)[j_{\text{ch}}] \). The only places in the listing at which \( L'_{i}(A)[j] \) and \( U'_{i}(A)[j] \) may be modified are lines 13, 14, 23, 27, and 28. Let us devise Hoare rules as preconditions to these lines.

The preconditions of the assignment in line 13 are as follows. First, the \( \$ \) invariant must hold, as it holds in the whole listing starting from line 10. Second, the copying variable must be set to true; otherwise, the \( i \) branch in line 12 that contains the assignments is not entered. Third, from the main while loop condition in line 10, \( i < \min \left( \text{len}(L'_{i-\varepsilon}(A)), \text{len}(L'_{i-\varepsilon}(B)) \right) \); otherwise, the while loop would have finished.

If we combine these three preconditions we get:

for \( L'_{i}(A)[i] \) to be modified in line 13 there must exist \( i_{\text{cop}} \) such that

\[
\begin{align*}
\min(k, \text{len}(L'_{i-\varepsilon}(B))) & \leq i_{\text{cop}} \quad \text{ and } \\
i_{\text{cop}} & < i \quad \text{ and } \\
L'_{i-\varepsilon}(A)[i_{\text{cop}}] & = L'_{i-\varepsilon}(B)[i_{\text{cop}}] \quad \text{ and } \\
U'_{i-\varepsilon}(A)[i_{\text{cop}}] & < U'_{i-\varepsilon}(B)[i_{\text{cop}}] \quad \text{ and } \\
\text{for all } i_{\text{cop}} < j < i & \left( L'_{i-\varepsilon}(A)[j] \neq L'_{i-\varepsilon}(B)[j] \right)
\end{align*}
\]
Therefore, for \( L_r(A)[i] \) to be modified in line 13, there must exist \( i_{\text{cop}} \) that, in particular, satisfies: \( \min(k, \text{len}(L_{r' - \varepsilon}(B))) \leq i_{\text{cop}} < i \). This means that if we have \( \text{len}(L_{r' - \varepsilon}(B)) \leq k \), then we also have \( \text{len}(L_{r' - \varepsilon}(B)) < i \), but from the third precondition, we have \( \text{len}(L_{r' - \varepsilon}(B)) > i \), hence we must have \( \text{len}(L_{r' - \varepsilon}(B)) > k \).

Therefore, for \( L_r(A)[i] \) to be modified in line 13, there must exist \( i_{\text{cop}} \) that, in particular, satisfies: \( k \leq i_{\text{cop}} < i \). This means that the level, \( i \), at which the label of \( A \) is modified is greater than \( k \), that is, \( i > k \). In other words, there cannot be any \( 0 \leq j_{\text{ch}} = i < k \) such that \( L_r(A)[j_{\text{ch}}] \neq L_r(A)[j_{\text{ch}}] \). Overall, the assignment in line 13 cannot invalidate the lemma.

The preconditions of the assignment in line 14 are the same as for line 13. Therefore again, for \( U_r(A)[i - 1] \) to be modified in line 14, there must exist \( i_{\text{cop}} \) that, in particular, satisfies: \( k \leq i_{\text{cop}} < i \). This means that the level, \( i - 1 \) at which the update vector of \( A \) is modified is greater than or equal to \( k \), that is, \( i - 1 \geq k \).

In other words, there cannot be any \( 0 \leq j_{\text{ch}} = i - 1 < k \) such that \( U_r(A)[j_{\text{ch}}] \neq U_r(A)[j_{\text{ch}}] \). Overall, the assignment in line 14 cannot invalidate the lemma either.

The preconditions of the assignment in line 23 are as follows. First, the \( \varepsilon \) invariant must hold, as it holds in the whole listing starting from line 10. Second, the \textit{copying} variable must be set to \textit{true}; otherwise, the \texttt{if} branch in line 22 that contains the assignments is not entered. Third, from the negation of the main \texttt{while} loop condition in line 10, \( i \geq \min\left(\text{len}(L_{r' - \varepsilon}(A)), \text{len}(L_{r' - \varepsilon}(B))\right) \); otherwise, the \texttt{while} loop cannot have finished.

If we combine these three preconditions we get:

for \( U_r(A)[i - 1] \) to be modified in line 23 there must exist \( i_{\text{cop}} \) such that

\[
\min(k, \text{len}(L_{r' - \varepsilon}(B))) \leq i_{\text{cop}} \quad \text{and} \quad i_{\text{cop}} < \min\left(\text{len}(L_{r' - \varepsilon}(A)), \text{len}(L_{r' - \varepsilon}(B))\right) \leq i
\]

\[
L_{r' - \varepsilon}(A)[i_{\text{cop}}] = L_{r' - \varepsilon}(B)[i_{\text{cop}}] \quad \text{and} \quad U_{r' - \varepsilon}(A)[i_{\text{cop}}] < U_{r' - \varepsilon}(B)[i_{\text{cop}}]
\]

\[
\text{for all } i_{\text{cop}} < j < \min\left(\text{len}(L_{r' - \varepsilon}(A)), \text{len}(L_{r' - \varepsilon}(B))\right)
\]

\[
(L_{r' - \varepsilon}(A)[j] \neq L_{r' - \varepsilon}(B)[j])
\]

Therefore, for \( U_r(A)[i - 1] \) to be modified in line 23, there must exist \( i_{\text{cop}} \) that, in particular, satisfies: \( \min(k, \text{len}(L_{r' - \varepsilon}(B))) \leq i_{\text{cop}} < \min\left(\text{len}(L_{r' - \varepsilon}(A)), \text{len}(L_{r' - \varepsilon}(B))\right) \leq i \). Let us again analyze this condition a bit. From the lemma assumption, we know that \( \text{len}(L_{r' - \varepsilon}(A)) \geq k \).

Therefore, if \( \text{len}(L_{r' - \varepsilon}(B)) \leq k \), then \( i_{\text{cop}} \) would have to in particular satisfy: \( \text{len}(L_{r' - \varepsilon}(B)) \leq i_{\text{cop}} < \text{len}(L_{r' - \varepsilon}(B)) \). Such an \( i_{\text{cop}} \) does not exist, hence we must consider \( \text{len}(L_{r' - \varepsilon}(B)) \geq k \). Given this, for \( U_r(A)[i - 1] \) to be modified
in line 23, there must exist $i_{cop}$ that, in particular, satisfies: $k \leq i_{cop} < \min\left(\text{len}(L_{t-e}(A))\right) \leq i$. This, however, means that the level, $i - 1$ at which the update vector of $A$ is modified in line 23 is greater than or equal to $k$, that is, $i - 1 \geq k$. In other words, there cannot be any $0 \leq j_{ch} = i - 1 < k$ such that $U_t(A)[j_{ch}] \neq U_t(A)[j_{ch}]$. Overall, the assignment in line 23 cannot invalidate the lemma either.

The preconditions of the assignments in lines 27 and 28 are the same as of the assignment in line 23. Therefore first, the level, $i$, at which the label of $A$ is modified in line 27 is greater than $k$, that is, $i > k$. In other words, there cannot be any $0 \leq j_{ch} = i < k$ such that $L_t(A)[j_{ch}] \neq L_t(A)[j_{ch}]$. Second, the level, $i - 1$ at which the update vector of $A$ is modified in line 28 is greater than or equal to $k$, that is, $i - 1 \geq k$. In other words, there cannot be any $0 \leq j_{ch} = i - 1 < k$ such that $U_t(A)[j_{ch}] \neq U_t(A)[j_{ch}]$. Overall, the assignments in lines 27 and 28 cannot invalidate the lemma either.

All in all, neither of the assignments results in a situation in which for some $0 \leq j_{ch} < k$ we have $L_t(A)[j_{ch}] \neq L_t(A)[j_{ch}]$ or $U_t(A)[j_{ch}] \neq U_t(A)[j_{ch}]$ — CONTRADICTION. This means that the second part of the lemma will also always hold after the label combine operation, that is, for all $0 \leq j < k$ we have $L_t(A)[j] = L_t(A)[j]$ and $U_t(A)[j] = U_t(A)[j]$.

Therefore, overall, the lemma holds after the label combination operation executed when node $A$ with a $k$-converged label receives a heartbeat message from node $B$ at time $t'$. Since node $B$, time $t'$, and level $k$ were chosen arbitrarily, no label combination operation at any time $t' \geq t$ can invalidate the lemma. Since the label combination is the only operation executed by node $A$ in a quiescent network, the lemma holds for node $A$ and any time $t' \geq t$. Since node $A$ and time $t$ were also chosen arbitrarily, the lemma holds in general. 

What the above lemma means is that in a quiescent network, once the prefixes of the current label and update vector of a node contain the freshest hierarchy membership information, these prefixes will not be modified. Therefore, considering also the proof of Lemma A.2 in a quiescent network, once the prefix of the current label of a node converged to the prefix of the correct label the node should have, the prefix will never diverge.

The next question to answer is thus: how do prefixes of the current label of a node converge to the corresponding prefixes of the correct label. The intuition from the previous proofs suggests that this happens in the label combine operation. We will support this intuition with the following lemma.

**Lemma A.5** Suppose that in a quiescent network some arbitrary node $A$ receives
a heartbeat message from some other arbitrary node B. Let us denote the current label and update vector of node A after the message reception as \( L_t'(A) \) and \( U_t'(A) \), respectively, and immediately before the message reception as \( L_{t-\varepsilon}(A) \) and \( U_{t-\varepsilon}(A) \), respectively. Likewise, \( L_t'(B) \), \( U_t'(B) \), \( L_{t-\varepsilon}(B) \), and \( U_{t-\varepsilon}(B) \) denote the corresponding data for node B that broadcast the heartbeat message.

Assume that for some arbitrary level \( k < \min \left( \text{len} \left( L_{t-\varepsilon}(A) \right), \text{len} \left( L_{t-\varepsilon}(B) \right) \right) \), we have:

1. for all \( 0 \leq j < k \), \( U_{t-\varepsilon}(A)[j] = U_{t-\varepsilon}(L_{t-\varepsilon}(A))[j] \), and
2. for all \( 0 \leq j \leq k \), \( U_{t-\varepsilon}(B)[j] = U_{t-\varepsilon}(L_{t-\varepsilon}(B))[j] \), and
3. \( L_{t-\varepsilon}(A)[k] = L_{t-\varepsilon}(B)[k] \), and
4. \( U_{t-\varepsilon}(A)[k] < U_{t-\varepsilon}(B)[k] \).

After the reception we will have, among others:

1. for all \( 0 \leq j \leq k \), \( U_t'(A)[j] = U_t'(L_t'(A))[j] \), and
2. if \( \text{len} \left( L_{t-\varepsilon}(B) \right) = k + 1 \), then \( \text{len} \left( L_t'(A) \right) = k + 1 \), else \( \text{len} \left( L_t'(A) \right) > k + 1 \) and \( L_t'(A)[k + 1] = L_{t-\varepsilon}(B)[k + 1] \).

**Proof:** Like before, we will perform the proof using Hoare logic on the pseudo-code of the label combine operation, which is the only operation executed on the label of a node during message reception. Therefore, like before let us now rewrite the label combination listing from page 94 with the notation from the lemma.
i \gets 0;
\text{copying} \gets \text{false};
\text{while } i < \min(\text{len}(L'_{\ell-e}(A)), \text{len}(L'_{\ell-e}(B))) \text{ do begin }
\text{if } \text{copying} \text{ then begin }
L'_{\ell}(A)[i] \leftarrow L'_{\ell-e}(B)[i];
U'_{\ell}(A)[i] \leftarrow U'_{\ell-e}(B)[i];
\text{end; }
\text{if } L'_{\ell-e}(A)[i] = L'_{\ell-e}(B)[i] \text{ then begin }
\text{if } U'_{\ell-e}(A)[i] \leq U'_{\ell-e}(B)[i] \text{ then } \text{copying} \gets \text{true; }
\text{else } \text{copying} \gets \text{false; }
\text{end; }
i \gets i + 1;
\text{end; }
\text{if } \text{copying} \text{ then begin }
U'_{\ell}(A)[i] \leftarrow U'_{\ell-e}(B)[i];
\text{len}(L'_{\ell}(A)) \leftarrow \text{len}(L'_{\ell-e}(B));
\text{len}(U'_{\ell}(A)) \leftarrow \text{len}(U'_{\ell-e}(B));
\text{while } i < \text{len}(L'_{\ell}(A)) \text{ do begin }
L'_{\ell}(A)[i] \leftarrow L'_{\ell-e}(B)[i];
U'_{\ell}(A)[i] \leftarrow U'_{\ell-e}(B)[i];
i \gets i + 1;
\text{end; }
\text{end; }
\{L'_{\ell}(B) = L'_{\ell-e}(B) \text{ and } U'_{\ell}(B) = U'_{\ell-e}(B)\}
\ldots

Going again through the whole listing and showing that subsequent properties of the lemma hold is extremely laborious and would waste many pages. Therefore, instead, below some logical statements resulting from a similar process like in the two previous proofs are given; the process of formally proving these statements using Hoare logic is thus left to the reader. The statements below precisely define how the node labels and update vectors can change during the label combination operation. We will use these statements to prove the lemma.

Statement 1.: the assignment in line 24, which changes the length of the label of node A, \(\text{len}(L'_{\ell}(A)) \leftarrow \text{len}(L'_{\ell-e}(B))\), is executed if and only if there exists some \(i'\), such that:

1. \(0 \leq i' < \min(\text{len}(L'_{\ell-e}(A)), \text{len}(L'_{\ell-e}(B)))\), and
2. \(L'_{\ell-e}(A)[i'] = L'_{\ell-e}(B)[i']\), and
3. \(U'_{\ell-e}(A)[i'] < L'_{\ell-e}(B)[i']\), and
4. for all \(i' < j < \min(\text{len}(L'_{\ell-e}(A)), \text{len}(L'_{\ell-e}(B)))\), \(L'_{\ell-e}(A)[j] \neq L'_{\ell-e}(B)[j]\);

otherwise, at the end of the label combination operation, \(\text{len}(L'_{\ell}(A)) = \text{len}(L'_{\ell-e}(A))\).
Statement 2.: the assignments in lines 13 and 27, which change the i-th element in the label of A, \( L_{t}^{\prime}(A)[i] \leftarrow L_{t}-\varepsilon(B)[i] \), are executed for those \( 0 \leq i < \text{len}(L_{t}-\varepsilon(B)) \), for which there exists some \( i' \), such that

1. \( 0 \leq i' < \text{min}(i, \text{len}(L_{t}-\varepsilon(A))) \), and
2. \( L_{t}-\varepsilon(A)[i'] = L_{t}-\varepsilon(B)[i'] \), and
3. \( U_{t}-\varepsilon(A)[i'] < U_{t}-\varepsilon(B)[i'] \), and
4. for all \( i' < j < \text{min}(i, \text{len}(L_{t}-\varepsilon(A))) \), \( L_{t}-\varepsilon(A)[j] \neq L_{t}-\varepsilon(B)[j] \);

otherwise, at the end of the label combination operation, \( L_{t}(A)[i] = L_{t}-\varepsilon(A)[i] \).

Statement 3.: the assignments in lines 14, 23, and 27, which change the i-th element in the update vector of A, \( U_{t}(A)[i] \leftarrow U_{t}-\varepsilon(B)[i] \), are executed for those \( 0 \leq i < \text{len}(L_{t}-\varepsilon(B)) \), for which there exists some \( i' \), such that

1. \( 0 \leq i' \leq \text{min}(i, \text{len}(L_{t}-\varepsilon(A))) - 1 \), and
2. \( L_{t}-\varepsilon(A)[i'] = L_{t}-\varepsilon(B)[i'] \), and
3. \( U_{t}-\varepsilon(A)[i'] < U_{t}-\varepsilon(B)[i'] \), and
4. for all \( i' < j < \text{min}(i, \text{len}(L_{t}-\varepsilon(A))) - 1 \), \( L_{t}-\varepsilon(A)[j] \neq L_{t}-\varepsilon(B)[j] \);

otherwise, at the end of the label combination operation, \( U_{t}(A)[i] = U_{t}-\varepsilon(A)[i] \).

We have to show that after the label combination operation we will have:

1. for all \( 0 \leq j < k \), \( U_{t}(A)[j] = U_{t}(L_{t}(A)[j]) [j] \), and
2. if \( \text{len}(L_{t}-\varepsilon(B)) = k + 1 \), then \( \text{len}(L_{t}(A)) = k + 1 \), else \( \text{len}(L_{t}(A)) > k + 1 \) and \( L_{t}(A)[k+1] = L_{t}-\varepsilon(B)[k+1] \).

To show the first property, let us observe that before the label combination operation for all \( 0 \leq j < k \), \( U_{t}(A)[j] = U_{t}(L_{t}-\varepsilon(A)[j]) [j] \), that is, before the operation A has the freshest hierarchy membership information at all levels below k. Therefore, from Lemma A.4 for all \( 0 \leq j < k \), we have \( U_{t}(A)[j] = U_{t}(L_{t}(A)[j]) [j] \), that is, after the label combination operation A also has the freshest hierarchy membership information at all levels below k. In other words, we just have to show that \( U_{t}(A)[k] = U_{t}(L_{t}(A)[k]) [k] \), that is, after the label combination operation, A will have the freshest hierarchy membership information also at level k.

\[1\]Actually, in the code, we sometimes have \( i-1 \) instead of \( i \). However, the condition is rephrased such that it is true for all three assignments.
Combining the fact that for all $0 \leq j < k$, $U_{t'-\varepsilon}(A)[j] = U_{t'-\varepsilon}(L_{t'-\varepsilon}(A)[j])[j]$ with Invariant A.3 on page 249, we know that for any $0 \leq j < k$, if $L_{t'-\varepsilon}(A)[j] = L_{t'-\varepsilon}(B)[j]$, then $U_{t'-\varepsilon}(A)[j] \geq U_{t'-\varepsilon}(B)[j]$. Therefore, from Statement 2, we get $L_{t'}(A)[k] = L_{t'-\varepsilon}(A)[k]$, that is, the label combination operation does not change the current label of node $A$ at level $k$.

Let us also observe that before the label combination operation we have $L_{t'-\varepsilon}(A)[k] = L_{t'-\varepsilon}(B)[k]$ and $U_{t'-\varepsilon}(A)[k] < U_{t'-\varepsilon}(B)[k]$, and moreover, $U_{t'-\varepsilon}(B)[k] = U_{t'-\varepsilon}(L_{t'-\varepsilon}(B)[k])[k]$, that is, node $B$ belongs to the same level-$k$ cluster as node $A$ and before the label combination operation it has the freshest information on the membership of this level-$k$ cluster in the hierarchy. In other words, there exists $i' = k$, such that $0 \leq i' < \min(k, \text{len}(L_{t'-\varepsilon}(A))) - 1$, and $L_{t'-\varepsilon}(A)[i'] = L_{t'-\varepsilon}(B)[i']$, and $U_{t'-\varepsilon}(A)[i'] < U_{t'-\varepsilon}(B)[i']$, and for all $i' < j \leq \min(k, \text{len}(L_{t'-\varepsilon}(A))) - 1$, $L_{t'-\varepsilon}(A)[j] \neq L_{t'-\varepsilon}(B)[j]$. Consequently, from Statement 3., we know that $U_{t'}(A)[k] = U_{t'-\varepsilon}(B)[k]$, that is, the update vector of node $A$ at level $k$ changes to the fresher value, which node $B$ has at this level. Since we know that $L_{t'}(A)[k] = L_{t'-\varepsilon}(B)[k]$ and $U_{t'-\varepsilon}(B)[k] = U_{t'-\varepsilon}(L_{t'-\varepsilon}(B)[k])[k]$ and $U_{t'}(A)[k] = U_{t'-\varepsilon}(B)[k]$, we know that $U_{t'}(A)[k] = U_{t'}(L_{t'}(A)[k])[k]$, which proves the first property.

To prove the second property, let us first assume that $\text{len}(L_{t'-\varepsilon}(B)) = k + 1$. In this case, when repeating the reasoning in the above paragraph, we learn that there exists $i' = k$, such that $0 \leq i' < \min(\text{len}(L_{t'-\varepsilon}(A)), \text{len}(L_{t'-\varepsilon}(B))) = k + 1$, and $L_{t'-\varepsilon}(A)[i'] = L_{t'-\varepsilon}(B)[i']$, and $U_{t'-\varepsilon}(A)[i'] < U_{t'-\varepsilon}(B)[i']$, and for all $i' < j < \min(\text{len}(L_{t'-\varepsilon}(A)), \text{len}(L_{t'-\varepsilon}(B))) = k + 1, L_{t'-\varepsilon}(A)[j] \neq L_{t'-\varepsilon}(B)[j]$. Therefore, from Statement 1., we know that $\text{len}(L_{t'}(A)) = \text{len}(L_{t'-\varepsilon}(B))$, that is, the length of the label of node $A$ is changed to the length of the label of $B$ as a result of the label combination operation. In other words, $\text{len}(L_{t'}(A)) = k + 1$, that is, the second property holds if $\text{len}(L_{t'-\varepsilon}(B)) = k + 1$.

Let us thus assume that $\text{len}(L_{t'-\varepsilon}(B)) > k + 1$. If $\text{len}(L_{t'-\varepsilon}(A)) = k + 1$ then again there exists $i' = k$, such that $0 \leq i' < \min(\text{len}(L_{t'-\varepsilon}(A)), \text{len}(L_{t'-\varepsilon}(B))) = k + 1$, and $L_{t'-\varepsilon}(A)[i'] = L_{t'-\varepsilon}(B)[i']$, and $U_{t'-\varepsilon}(A)[i'] < L_{t'-\varepsilon}(B)[i']$, and for all $i' < j < \min(\text{len}(L_{t'-\varepsilon}(A)), \text{len}(L_{t'-\varepsilon}(B))) = k + 1, L_{t'-\varepsilon}(A)[j] \neq L_{t'-\varepsilon}(B)[j]$. Therefore, again from Statement 1., we know that $\text{len}(L_{t'}(A)) = \text{len}(L_{t'-\varepsilon}(B))$, that is, the length of the label of node $A$ is changed to the length of the label of $B$ as a result of the label combination operation. In effect, $\text{len}(L_{t'}(A)) > k + 1$. If, in turn, $\text{len}(L_{t'-\varepsilon}(A)) > k + 1$, then irrespective of whether the length of the label of node $A$ is changed to the length of the label of $B$ or not, we have $\text{len}(L_{t'}(A)) > k + 1$.

Let us thus analyze $L_{t'}(A)[k + 1]$. Again, we know that $L_{t'-\varepsilon}(A)[k] = L_{t'-\varepsilon}(B)[k]$ and $U_{t'-\varepsilon}(A)[k] < U_{t'-\varepsilon}(B)[k]$. In other words, there exists $i' = k$, such
A.6. Proof of Lemma 3.6

that \(0 \leq i' < \min(k+1, \text{len}(L_{t'-\varepsilon}(A))) = k+1\), and \(L_{t'-\varepsilon}(A)[i'] = L_{t'-\varepsilon}(B)[i']\), and \(U_{t'-\varepsilon}(A)[i'] < U_{t'-\varepsilon}(B)[i']\), and for all \(i' < j < \min(k+1, \text{len}(L_{t'-\varepsilon}(A))) = k+1\), \(L_{t'-\varepsilon}(A)[j] \neq L_{t'-\varepsilon}(B)[j]\). Therefore, from from Statement 2. we get 
\(L_{t'}(A)[k+1] = L_{t'}(B)[k+1]\), that is, the label combination operation changes the current label of node \(A\) at level \(k+1\) to the value of the label of \(B\) at this level. In other words, when \(\text{len}(L_{t'-\varepsilon}(B)) > k+1\), the second property holds as well.

Proving the two properties proves the lemma.

From the above lemma, we know that, if a node the label of which is \(k\)-converged (i.e., contains the freshest hierarchy membership information at all levels lower than \(k\)) receives a heartbeat message from a neighbor the label of which is \(k+1\)-converged, and the neighbor belongs to the same level-\(k\) cluster as the node, then, as a result, the label of the node itself becomes \(k+1\)-converged. In other words, we know how the freshest membership information propagates over one hop.

We will now show that the freshest hierarchy membership information actually propagates over multiple hops, that is, that a \(k\)-converged node that initially may not have any \(k+1\)-converged neighbors eventually becomes \(k+1\)-converged. More specifically, we will prove the following supporting lemma.

**Lemma A.6** Consider a quiescent network in which the cluster hierarchy reflected in the correct labels of all nodes satisfies Properties 1-4. In such a network, if there exists time \(t_k\) at which the labels of all nodes are \(k\)-converged, that is, for any node \(A\) and any \(0 \leq j < \min\left(k, \text{len}(L_{t_k}(A))\right)\), \(U_{t_k}(A)[j] = U_{t_k}(L_{t_k}(A)[j])[j]\), then there exists time \(t_{k+1}\) at which the labels of all nodes are \(k+1\)-converged, that is, for any node \(A\) and any \(0 \leq j < \min\left(k+1, \text{len}(L_{t_{k+1}}(A))\right)\), \(U_{t_{k+1}}(A)[j] = U_{t_{k+1}}(L_{t_{k+1}}(A)[j])[j]\).

**Proof:** To prove the lemma we will first show that, in a network that is \(k\)-converged at time \(t_k\), there exists at least one node that is \(k+1\)-converged at time \(t_k\). We will then show that by gossiping heartbeat messages with each other, and thus, executing the label combination operation, more and more nodes become \(k+1\)-converged, such that eventually there exists time \(t_{k+1}\) at which all the nodes are \(k+1\)-converged.

To this end, let us take an arbitrary level \(k\) and let us assume that a network satisfying the assumptions of the lemma becomes \(k\)-converged at time \(t_k\), that is, for any node \(A\) and any \(0 \leq j < \min\left(k, \text{len}(L_{t_k}(A))\right)\), \(U_{t_k}(A)[j] = U_{t_k}(L_{t_k}(A)[j])[j]\).
Let us take an arbitrary node $A$. If $k \geq \text{len}(L_k(A))$, then $A$ is already $k+1$-converged at time $t_k$ because for all $0 \leq j < \min(k+1, \text{len}(L_k(A))) = \min(k, \text{len}(L_k(A))) = \text{len}(L_k(A))$, $U_k(A)[j] = U_{t_k}(L_k(A)[j])[j]$. Therefore, we assume that $k < \text{len}(L_k(A))$.

This means that $L_k(A)[k] = V$ for some node $V$. From the proof of Lemma A.2 and the fact that for all $0 \leq j < k$, $U_k(A)[j] = U_{t_k}(L_k(A)[j])[j]$, we know that for all $0 \leq j < k + 1$, $L_k(A)[j] = L_{*k}(A)[j]$, hence in particular, $L_{*k}(A)[k] = V$. Since from the lemma assumptions, the hierarchy reflected in the correct labels of all nodes satisfies the four properties, for all $0 \leq j < k + 1$ we must have, $L_{*k}(V)[j] = V$, which means that for all $0 \leq j < k + 1$, $L_{*k}(A)[j] = V$. This, however, means that $U_{t_k}(V)[k] = U_{t_k}(L_{*k}(V)[k])[k]$, that is, node $V$ is already $k+1$-converged at time $t_k$. The fact that $V$ is already $k+1$-converged at time $t_k$ has a trivial intuitive explanation: the head of a level-$k$ cluster has the freshest information about the membership of this cluster at level $k+1$ in the hierarchy.

We will thus denote the $k+1$-convergence time of node $A$ as $t_{k+1}^A$; we already know that $t_{k+1}^A = t_k$.

The hierarchy reflected in the correct labels of all nodes satisfies the four properties. Therefore, in particular, Lemma A.1 holds, that is, there exists a path, $\pi = V = A_0, A_1, \ldots, A_{d-1}, A_d = A >$, such that for all $0 \leq i < d$, $A_i$ and $A_{i+1}$ are neighbors and for all $0 \leq i \leq d$, $L_{*k}(A_i)[k] = V$. Since the network is $k$-converged, this also means that for all $0 \leq i \leq d$, $L_{*k}(A_i)[k] = V$.

We will prove by induction that in at most $i$ gossip rounds, node $A_i$ becomes $k+1$-converged. In other words, if $T$ denotes the round length, then we can be certain that at time $t_{k+1}^A = t_k + i \cdot T$, for all $0 \leq j < k + 1$, $U_{t_k}^A(A_i)[j] = U_{t_k}^A(L_{*k}(A_i)[j])[j]$. 

\[ \text{T = length of gossip round} \]
Basis: \( i = 0 \). In this case, \( A_i = V \). We know that \( V \) becomes \( k+1 \)-converged at time \( t'_{k+1} = t_k = t_k + 0 \cdot T = t_k + i \cdot T \). Therefore, in at most \( i = 0 \) rounds, node \( A_0 \) becomes \( k+1 \)-converged.

Inductive step: \( i > 0 \). From the inductive assumption, node \( A_{i-1} \) becomes \( k+1 \)-converged at time \( t'_{k+1} \leq t_k + (i-1) \cdot T \). The gossiping round lasts \( T \) time units. Moreover, let us assume that processing a received heartbeat message lasts \( \varepsilon \rightarrow 0 \) time units. Therefore, within at most \( T - \varepsilon \) time units from \( t'_{k+1} \) — at some time \( t'_{k+1} - \varepsilon \leq t'_{k+1} + T - \varepsilon \) — node \( A_{i-1} \) will broadcast its heartbeat message. Assuming no packet loss, the message will be received by node \( A_i \) because \( A_{i-1} \) and \( A_i \) are neighbors.

If node \( A_j \) is already \( k+1 \)-converged at time \( t' - \varepsilon \), that is, \( 0 \leq j < k+1, U_{t' - \varepsilon}(A_i)[j] = U_{t' - \varepsilon}(L_{t' - \varepsilon}(A_i)[j])[j] \), then we have \( t'_{k+1} = t' - \varepsilon \leq t'_{k+1} + T - \varepsilon \leq t_k + (i-1) \cdot T + T - \varepsilon = t_k + i \cdot T \). Therefore, let us assume that at time \( t' - \varepsilon \), node \( A_j \) is not yet \( k+1 \)-converged.

However, since \( A \) is \( k \)-converged, we know that only \( U_{t'_{k+1} - \varepsilon}(A_i)[k] \neq U_{t'_{k+1} - \varepsilon}(L_{t'_{k+1} - \varepsilon}(A_i)[k])[k] \), more specifically, \( U_{t'_{k+1} - \varepsilon}(A_i)[k] < U_{t'_{k+1} - \varepsilon}(L_{t'_{k+1} - \varepsilon}(A_i)[k])[k] \).

Overall, we know that at time \( t' - \varepsilon \):

- for all \( 0 \leq j < k, U_{t' - \varepsilon}(A_i)[j] = U_{t' - \varepsilon}(L_{t' - \varepsilon}(A_i)[j])[j] \), and
- for all \( 0 \leq j \leq k, U_{t' - \varepsilon}(A_{i-1})[j] = U_{t' - \varepsilon}(L_{t' - \varepsilon}(A_{i-1})[j])[j] \), and
- \( L_{t' - \varepsilon}(A_i)[k] = L_{t' - \varepsilon}(A_{i-1})[k] = V \), and
- \( U_{t' - \varepsilon}(A_j)[k] < U_{t' - \varepsilon}(A_{i-1})[k] \).

From Lemma A.5, we know that after receiving the heartbeat message from node \( A_{i-1} \), at time \( t' \), node \( A_i \) will become \( k+1 \)-converged, that is, for all \( 0 \leq j < k+1, U_{t'}(A_i)[j] = U_{t'}(L_{t'}(A_i)[j])[j] \). Consequently, \( t'_{k+1} = t' \leq t'_{k+1} + T - \varepsilon \leq t_k + (i-1) \cdot T + T = t_k + i \cdot T \). In other words, in at most \( i \) rounds, node \( A_i \) will be \( k+1 \) converged.

By applying mathematical induction to the basis and the inductive step, we prove that any node \( A_i \) on path \( \pi \) will be \( k+1 \)-converged within at most \( i \) rounds. Since node \( A \), node \( V \), and path \( \pi \) were chosen arbitrarily, for any node, \( B \), there exists time \( t_{k+1}^B \geq t_k \) at which \( B \) will become \( k+1 \) converged. Since the node population is finite, there exists time \( t_{k+1} = \max_{t_{k+1} \in \text{all nodes}}(t_{k+1}^B) \geq t_k \) at which the labels of all nodes will be \( k+1 \)-converged. This proves the lemma. ■

Since we know how updates propagate in the network, we are now ready to prove the main lemma.
The Lemma: Consider a quiescent network, that is, a network in which, starting from some time \( t_q \), there are no changes in the connectivity graph reflected in the neighbor relation and no label extension or label cut operations are executed by any node. In such a network, if the cluster hierarchy reflected in the correct labels of all nodes satisfies Properties \([\text{I} \text{H}]\) then there exists some time \( t_c \geq t_q \), such that for any node \( A \) and any time \( t \geq t_c \), \( L_t(A) = L_t^*(A) \). In other words, in a quiescent network the current labels maintained locally by the nodes eventually become consistent with the correct labels the nodes should have.

Proof: To prove the lemma let us consider a \( k \)-element prefix of a label. We will prove by induction that for any \( k \), there exists time \( t_k \) such that for any time \( t \geq t_k \) the labels of all nodes are \( k \)-converged, that is, for any node \( A \) and any \( 0 \leq j < \min \left( k, \text{len} \left( L_t(A) \right) \right) \), \( U_t(A)[j] = U_t \left( L_t(A)[j] \right)[j] \).

**Basis:** \( k = 1 \). Let us take an arbitrary node, \( A \). From Invariant \([\text{A.I}]\) on page 248, for any \( t \), we have \( \text{len} \left( L_t(A) \right) > 1 \) and \( L_t(A)[0] = A \), thus also, \( U_t(A)[0] = U_t \left( L_t(A)[0] \right)[0] \). In particular, \( U_{t_q}(A)[0] = U_{t_q} \left( L_{t_q}(A)[0] \right)[0] \), hence \( t_q^0 = t_q \). Since node \( A \) was chosen arbitrarily, we can take \( t_1 = t_q \). In other words, the lemma holds for the basis.

**Inductive step:** \( k > 1 \). From the inductive assumption, there exists \( t_{k-1} \geq t_q \) such that for any time \( t \geq t_{k-1} \), any node \( A \), and any \( 0 \leq j < \min \left( k - 1, \text{len} \left( L_t(A) \right) \right) \), \( U_t(A)[j] = U_t \left( L_t(A)[j] \right)[j] \). Therefore, from Lemma \([\text{A.6}]\) there exists time \( t_k \geq t_{k-1} \) at which, for any node \( A \) and any \( 0 \leq j < \min \left( k, \text{len} \left( L_{t_k}(A) \right) \right) \), \( U_{t_k}(A)[j] = U_{t_k} \left( L_{t_k}(A)[j] \right)[j] \). However, from Lemma \([\text{A.4}]\) for any time \( t \geq t_k \), for any node \( A \) and any \( 0 \leq j < \min \left( k, \text{len} \left( L_t(A) \right) \right) \), \( L_t(A)[j] = L_{t_k}(A)[j] \) and \( U_t(A)[j] = U_{t_k}(A)[j] \). Overall, there exists time \( t_k \geq t_q \), such that for any \( t \geq t_k \), any node \( A \), and any \( 0 \leq j < \min \left( k, \text{len} \left( L_t(A) \right) \right) \), \( U_t(A)[j] = U_t \left( L_t(A)[j] \right)[j] \). In other words, the lemma holds for the inductive step.

By applying mathematical induction to the basis and the inductive step, we prove that for any \( k \), there exists time \( t_k \geq t_q \) such that for any time \( t \geq t_k \) the labels of all nodes are \( k \)-converged, that is, for any node \( A \) and any \( 0 \leq j < \min \left( k, \text{len} \left( L_t(A) \right) \right) \), \( U_t(A)[j] = U_t \left( L_t(A)[j] \right)[j] \).

Since the hierarchy reflected in the correct labels of all nodes satisfies the four properties, in particular, Property \([\text{I} \text{H}]\) which states that the height of the hierarchy is finite, there exists \( H^* \), such that \( \text{len} \left( L_t^*(A) \right) \leq H^* + 1 \) for any node \( A \) and any
time $t \geq t_q$. Consequently, there exists $t_{\mathcal{H}+1} \geq t_q$ such that for any time $t \geq t_{\mathcal{H}+1}$ the labels of all nodes are fully converged, that is, for any node $A$ and any $0 \leq j < \text{len}(L_t(A))$, $U_t(A)[j] = U_t(L_t(A)[j])$. Therefore, from Lemma A.2, there exists $t_{c} = t_{\mathcal{H}+1} \geq t_q$ such that for any time $t \geq t_{c}$ and any node $A$, $L_t(A) = L^*_t(A)$. Since from Lemma A.3, for any $t \geq t_q$, $L^*_t(A) = L^*_{t_q}(A)$, we have there exists $t_{c} = t_{\mathcal{H}+1} \geq t_q$ such that for any time $t \geq t_{c}$ and any node $A$, $L_t(A) = L^*_{t_q}(A)$. In other words, there exists time $t_{c}$ at which the current labels of all nodes have converged to the correct labels the nodes should have. This proves the main lemma — Lemma 3.6.

A.7. PROOF OF LEMMA 3.7

**The Lemma:** Assume that the slot size, $r$, is longer than the number of rounds it takes to propagate information between the heads, $V$ and $U$, of two adjacent “top-level” clusters $C_i^V$ and $C_i^U$. In this case, with probability $\geq \frac{1}{4}$, $C_i^V$ will be able to join $C^i_{U+1}$ or vice versa.

**Proof:** Consider two arbitrary nodes, $V$ and $U$, that are heads of adjacent “top-level” clusters, $C_i^V$ and $C_i^U$, respectively. $V$ and $U$ have to potentially spawn level-$i+1$ clusters.

Let $r_V$ and $r_U$ denote the round in which $V$ and $U$ respectively choose their virtual slots, as described in Section 3.4.3. Note that this implies that in round $r_V$, $V$ has learned about $U$, and, similarly, in round $r_U$, $U$ has learned about $V$. Let the number of slots $S = 2$. Moreover, assume that the slot size, $r$, meets the requirements of the lemma, that is, it is longer than the number of rounds necessary to propagate information between $V$ and $U$.

There are the following four possible slot selection configurations, each obtained with probability $\frac{1}{4}$:

<table>
<thead>
<tr>
<th>$s_V$</th>
<th>$s_U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Without the loss of generality assume that $r_V \geq r_U$, that is, $V$ selects its slot in the same or later round than $U$. Consider configuration III, in which $V$ selects slot $s_V = 1$ and $U$ selects slot $s_U = 0$. Let $r^*_V = r_V + s_V \cdot r + 1$ denote the round in which $V$ potentially spawns cluster $C^i_{V+1}$, as specified by the algorithm. Likewise, let $r^*_U = r_U + s_U \cdot r + 1$ denote the round in which $U$ potentially spawns cluster $C^i_{U+1}$. We will show (by contradiction) that by the time it spawns cluster $C^i_{V+1}$,
node $V$ discovers that node $U$ has spawned $C_i^{i+1}$. Consequently, $V$ can make $C_V^i$ a subcluster of $C_U^{i+1}$, decreasing the number of clusters at level $i+1$.

To this end, assume that $V$ spawns $C_V^{i+1}$ in round $r_V^*$ and $U$ spawns $C_U^{i+1}$ in round $r_U^*$. Consider value $r_V^* - r_U^*$, which denotes how many rounds after node $U$ has spawned cluster $C_U^{i+1}$, node $V$ spawns cluster $C_V^{i+1}$.

From the above calculation $V$ spawns cluster $C_V^{i+1}$ at least $r$ rounds after $U$ has spawned cluster $C_U^{i+1}$. **CONTRADICTION**, because within at most $r$ rounds, $V$ would have learned that $U$ spawned $C_U^{i+1}$, and, consequently, would have made $C_V^i$ a subcluster of $C_U^{i+1}$. Therefore, with probability at least $\frac{1}{4}$, $C_V^i$ and $C_U^i$ will be subclusters of a common cluster $C_{V/U}^{i+1}$. Because $V$, $U$, $C_V^i$, and $C_U^i$ were chosen arbitrarily, the lemma holds for any head node at any level. In practice, the aforementioned probability can be higher than $\frac{1}{4}$.

---

**A.8. INvariants of Labels and Update Vectors**

This section describes some general invariants of labels and update vectors. The common notation is used to denote the labels and update vectors maintained by nodes. Symbol $S_t(A)$ is used to denote the state at time $t$ of the sequence number generator for label updates of performed by node $A$, that is, the value of variable `this.addrUpdSeqNo` of node $A$ in the listings from Chapter 3. Each invariant can be proved using Hoare logic on the code listings from Chapter 3, much like was done in the proof of Lemma 3.6 in Section A.6. More specifically, Listing 3.9 should be used for proving the invariants after initialization, Listing 3.7 for the label extension operation, Listing 3.8 for the label cut operation, and Listing 3.11 (or directly Listing 3.6) for the label combination operation. Although such proofs are straightforward, they are also laborious and mundane. For this reason, they are left to the reader.

**Invariant A.1** (The level-0 element of a label is always equal to the identifier of the node storing the label.): For any active node $A$ and any time $t$, $\text{len}(L_t(A)) \geq 1$ and $L_t(A)[0] = A$.  

---
Invariant A.2 (The head of a level-i cluster is also the head of all lower-level clusters.): For any active node A, any time t, and any level i, if $L_t(A)[i] = A$, then $L_t(A)[j] = A$ for all $0 \leq j \leq i$.

Invariant A.3 (The head of a cluster has the freshest information about the membership of the cluster in the hierarchy.): For any active nodes A and V, any time t, and any level i, if $L_t(A)[i] = V$ and $L_t(V)[i] = V$, then $U_t(A)[i] \leq U_t(V)[i]$.

Invariant A.4 (The information about the membership of a cluster in the hierarchy is consistent at two nodes if the freshness of the information is the same at these nodes.): For any active nodes A and V, any time t, and any level i, if $L_t(A)[i] = V$ and $L_t(V)[i] = V$ and $U_t(A)[i] = U_t(V)[i]$, then either:

- $\text{len}(L_t(A)) = \text{len}(L_t(V)) = i + 1$, or
- $\text{len}(L_t(A)) > i + 1$ and $\text{len}(L_t(V)) > i + 1$ and $L_t(A)[i+1] = L_t(V)[i+1]$.

Invariant A.5 (Stale cluster membership information does not override fresher information.): For any active node A, any time t and $t'$ such that $t \leq t'$, any level i, if $L_t(A)[i] = L_{t'}(A)[i]$, then $U_t(A)[i] \leq U_{t'}(A)[i]$.

Invariant A.6 (The membership information does not change unless it gets stale.): For any active node A, any time t and $t'$ such that $t \leq t'$, and any level i, if $L_t(A)[i] = L_{t'}(A)[i]$ and $U_t(A)[i] = U_{t'}(A)[i]$, then either:

- $\text{len}(L_t(A)) = \text{len}(L_{t'}(A)) = i + 1$, or
- $\text{len}(L_t(A)) > i + 1$ and $\text{len}(L_{t'}(A)) > i + 1$ and $L_t(A)[i+1] = L_{t'}(A)[i+1]$.

Invariant A.7 (A sequence number generator always generates the freshest sequence number.): For any active node A, any time t, and any level i, if $L_t(A)[i] = A$, then $U_t(A)[i] \leq S_t(A)$.
Appendix B

Experimental Testbed

In this appendix, one can find information on a 60-node indoor testbed, KonTest, which I have designed and deployed together with Albana Gabay at Vrije Universiteit Amsterdam. The appendix outlines the hardware architecture and the organization of the testbed. It also shows basic network properties, which were obtained through experiments. An extended version of this chapter can be found in an earlier technical report [51].

B.1. HARDWARE AND ORGANIZATION

The main hardware components constituting the testbed are presented in Table B.1. The testbed includes 60 TelosB sensor node modules [106] (see also Figure B.1 and Table B.2). Ten of the nodes have been purchased at Moteiv Corporation [2], each of those nodes contains a full sensor suite. The remaining

1 Albana Gabay, Ph.D. student, Department of Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081A, 1081 HV Amsterdam, the Netherlands, e-mail: agaba@cs.vu.nl.
2 http://www.moteiv.com/

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission of the author and/or a fee.

fifty nodes, in turn, come from Crossbow Technology Inc[^3] twenty of them are with and thirty are without the sensor suite.

![TelosB node with sensor suite](http://www.xbow.com/)

Table B.1: The main hardware components of the experimental testbed. *These are just the active components. The table does not include hundreds of meters of cabling for both USB and Ethernet.*

<table>
<thead>
<tr>
<th>Component</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>TelosB (Moteiv) w/ sensors</td>
<td>10</td>
</tr>
<tr>
<td>TelosB (Crossbow) w/ sensors</td>
<td>20</td>
</tr>
<tr>
<td>TelosB (Crossbow) w/o sensors</td>
<td>30</td>
</tr>
<tr>
<td>USB hubs</td>
<td>16</td>
</tr>
<tr>
<td>PCs</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure B.1: A TelosB node with the sensor suite (TMote Sky). *Visible in the photo are among others: two buttons, a USB chip, two light sensors, a temperature and relative humidity sensor, a radio chip, and an antenna. The MCU is located on the other side of the board.*

The nodes are located on the fourth floor of the southern P-wing and the western R-wing of the Faculty of Sciences of Vrije Universiteit Amsterdam. They are dispersed among six office rooms as depicted in Figure B.2. Such a distribution impacts wireless internode connectivity, as discussed in the next section, thereby providing a realistic network topology. In addition, it allows for collecting environmental data from different parts of the building: southern, eastern, and western.

To avoid troublesome battery-based power supply and to enable low-overhead node retasking and statistics reporting, each of the nodes is connected to a PC using a USB network. The USB network consists of a number of cables and...
**B.1. Hardware and Organization**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MODULE</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor Performance</td>
<td>16-bit RISC</td>
<td></td>
</tr>
<tr>
<td>Program Flash Memory</td>
<td>48K bytes</td>
<td></td>
</tr>
<tr>
<td>Measurement Serial Flash</td>
<td>1024K bytes</td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td>10K bytes</td>
<td></td>
</tr>
<tr>
<td>Configuration EEPROM</td>
<td>16K bytes</td>
<td></td>
</tr>
<tr>
<td>Serial Communications</td>
<td>UART</td>
<td>0-3V transmission levels</td>
</tr>
<tr>
<td>Analog to Digital Converter</td>
<td>12 bit ADC</td>
<td>8 channels, 0-3V input</td>
</tr>
<tr>
<td>Digital to Analog Converter</td>
<td>12 bit DAC</td>
<td>2 ports</td>
</tr>
<tr>
<td>Other Interfaces</td>
<td>Digital I/O, I2C, SPI</td>
<td></td>
</tr>
<tr>
<td>Current Draw</td>
<td>1.8 mA</td>
<td>Active mode</td>
</tr>
<tr>
<td></td>
<td>5.1 µA</td>
<td>Sleep mode</td>
</tr>
<tr>
<td><strong>RF TRANSCEIVER</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequency band</td>
<td>2400-2483.5 MHz</td>
<td>ISM band</td>
</tr>
<tr>
<td>Transmit (TX) data rate</td>
<td>250 kbps</td>
<td></td>
</tr>
<tr>
<td>RF power</td>
<td>-25 dBm to 0 dBm</td>
<td></td>
</tr>
<tr>
<td>Receive Sensitivity</td>
<td>-90/-94 dBm</td>
<td>minimal/typical</td>
</tr>
<tr>
<td>Outdoor Range</td>
<td>75 m to 100 m</td>
<td>Inverted-F antenna</td>
</tr>
<tr>
<td>Indoor Range</td>
<td>20 m to 30 m</td>
<td>Inverted-F antenna</td>
</tr>
<tr>
<td>Current Draw</td>
<td>23 mA</td>
<td>Receiver/Listen</td>
</tr>
<tr>
<td></td>
<td>21 mA</td>
<td>Transmit (at 0 dBm)</td>
</tr>
<tr>
<td></td>
<td>1 µA</td>
<td>Off</td>
</tr>
<tr>
<td><strong>SENSORS (Optional)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visible Light Sensor Range</td>
<td>320 nm to 730 nm</td>
<td>Hamamatsu S1087</td>
</tr>
<tr>
<td>Visible IR Sensor Range</td>
<td>320 nm to 1100 nm</td>
<td>Hamamatsu S1087-01</td>
</tr>
<tr>
<td>Humidity Sensor Range Resolution Accuracy</td>
<td>0-100% RH 0.03% RH ± 3.5% RH Absolute RH</td>
<td>Sensirion SHT11</td>
</tr>
<tr>
<td>Temperature Sensor Range</td>
<td>-40°C to 123.8°C 0.01°C ± 0.5°C at 25°C</td>
<td>Sensirion SHT11</td>
</tr>
<tr>
<td>Resolution Accuracy</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MECHANICAL</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Battery</td>
<td>2X AA batteries</td>
<td>Attached pack</td>
</tr>
<tr>
<td>User Interface</td>
<td>USB v1.1 or higher</td>
<td></td>
</tr>
<tr>
<td>Size (in) (mm)</td>
<td>2.55 x 1.24 x 0.24 65 x 31 x 6</td>
<td>w/o battery pack</td>
</tr>
<tr>
<td>Weight (oz) (grams)</td>
<td>0.8 23</td>
<td>w/o batteries</td>
</tr>
</tbody>
</table>

*Programmable in 1-MHZ steps, 5-MHZ steps for compliance with IEEE 802.15.4/D18-2003.

**Table B.2:** The specification of a TelosB sensor node. *TelosB is an ultra-low-power sensor node, widely used for experimental purposes. The specification is based on the datasheets published by Crossbow Technology, Inc.*

hubs connected to six Pentium III PCs, one PC per room. It is used only as a power supply and a reliable transport backbone for protocol statistics and control commands. The protocols evaluated on the testbed, in contrast, use the standard wireless communication. In this way, the interference between statistic gathering and protocol operation is minimized.
Figure B.2: The node placement in our testbed. *The nodes are located on the fourth floor the southern P-wing and the western R-wing of the Faculty of Sciences of Vrije Universiteit Amsterdam.*

B.2. BASIC NETWORK PROPERTIES

To obtain basic connectivity properties of our network, a simple TinyOS 2.0 application has been deployed.

B.2.1. Experimental Setup

The test application worked in periods. In the default configuration, each period lasted 30 seconds and the clocks of the nodes were not synchronized. In every period, each node broadcast a heartbeat message at a uniformly random moment.
within the period. A heartbeat message of a node consisted of the identifier of the node (2 bytes), a sequence number (4 bytes), and a “Hello world from Konrad Iwanicki!” string (34 bytes including the terminating null).

Nodes that heard the message transmission recorded the following metadata: the local time, the identifier and sequence number of the sender, and the information about the received signal for the message, as provided by the CC2420 radio chip in the form of the received signal strength indicator (RSSI) and the link quality indicator (LQI). The recorded message metadata were placed in a metadata queue.

At the end of every period, all records from the metadata queue of a node were transmitted over the USB network to the PC the node was connected to. The PC logged those metadata using a Java frontend of our application. Therefore, ultimately the log of each node contained fine-grained data specifying all the messages the node had received and the signal quality when receiving these messages. Equipped with such data, we were able to assess the wireless internode connectivity in our testbed.

We have conducted four deployments of the application, each lasting at least 24 hours. Each deployment varied in the node transmission power, which was set globally for all nodes. We used the following transmission power settings: $-25 \text{ dBm}$, $-15 \text{ dBm}$, $-5 \text{ dBm}$, and $0 \text{ dBm}$. This allowed us to investigate the impact of the transmission range on the network density and diameter. Since during the experiments the 5 nodes in room P4.46 were not deployed, we used only the remaining 55 nodes in other rooms.

### B.2.2. Experimental Results

For each pair of nodes, in each direction, we analyzed RSSI, LQI, and the packet loss rate (PLR), which was obtained by examining the sequence numbers of the received messages. In this way, considering for instance the average RSSI, we obtained a connectivity matrix in which a cell in row $i$ and column $j$ represents the average RSSI value for messages received by node $i$ from node $j$. Such sample matrices for the transmission power of $-25 \text{ dBm}$ are depicted in Figure B.3, Figure B.4, and Figure B.5.

These figures show inherent clustering between the nodes. Nodes in the same room are likely to be connected with a high-quality link (high RSSI and LQI values, and a low PLR value). There are obviously some exceptions, such as nodes 02 and 03 or nodes 48 and 45. In addition, some nodes in every room can communicate with some nodes in other rooms, which ultimately results in a single connected network. Finally, all three link quality metrics are strongly correlated,
Figure B.3: An RSSI matrix for the testbed (RF power of $-25$ dBm). A cell in row $i$ and column $j$ of the matrix represents the average RSSI value for messages received by node $i$ from node $j$. The matrix illustrates high clustering of RSSI values for nodes that are in the same office rooms.

again with an exception of a few links, which matches the results reported by other groups [118, 120, 138].

Based on the connectivity matrices, we also generated connectivity graphs. Each vertex in the graph corresponds to a node. There exists an edge between two vertices if the communication links between the nodes associated with these vertices meet certain requirements. For the RSSI metric, for instance, the RSSI value of the links in both directions must be at least $-90$ dBm. For the LQI metric, in turn, the LQI value of the links in both directions must be at least 95. Finally, for PLR metric, the packet loss rate of the links must be below 15%. These thresholds represent high-quality links and were obtained from the results reported by other groups [118, 120, 138].

For different transmission power levels and link quality metrics, we computed certain graph-theoretic properties of the connectivity graphs. Those properties included: the graph diameter (the maximal shortest path between all pairs of nodes), the network density (the degree of a node in the connectivity graph), and the clustering coefficient (the ratio of the actual number of links between the neighbors of a node to all the possible links between those neighbors). The results for different metrics and power levels are shown in Table B.3.

It can be observed that in all configurations the network is multi-hop with the
Figure B.4: An LQI matrix for the testbed (RF power of $-25$ dBm). A cell in row $i$ and column $j$ of the matrix represents the average LQI value for messages received by node $i$ from node $j$. The matrix illustrates high clustering of LQI values for nodes that are in the same office rooms. To a large extent, it also matches the RSSI matrix from Figure B.3.

The connectivity graph is highly clustered, as mentioned previously. High clustering is an inherent feature of wireless networks as, due to the limited radio range, two nodes that have a common neighbor are very likely to be each other’s neighbors as well [101].

To complete the picture, Figure B.6 and Figure B.7 show the distribution of node degree and path length for the PLR metric and the lowest transmission power.

The degree distribution (see Figure B.6) demonstrates that more than 50% of the nodes have at most 17 good quality neighbors. Yet, there exist some nodes that have more than 25 neighbors. In general, the node degree distribution is highly nonuniform and varies between 8 and 31 neighbors. Such nonuniformity is very common in real-world sensornet deployments.

The distribution of the path length (see Figure B.7) is more predictable. Slightly more than 30% of all 2970 paths are one hop. This is a direct consequence of the high clustering coefficient and the small scale of the testbed. More than 60% of the paths are at most two hops, and nearly 90% are at most
three hops. However, a relatively large fraction of paths has the same length as the network diameter. This implies that, to reach each other, many nodes must forward messages over the distance equal to the network diameter. Such long paths facilitate testing various routing protocols.

B.3. CONCLUSIONS

This appendix introduced our 60-node indoor sensornet testbed. It outlined the hardware architecture of the testbed and presented the basic properties of the internode connectivity graph. The graph has highly nonuniform node density and many paths reaching the network diameter of 4 to 5 hops. I believe that these properties of the testbed enable sound evaluation of sensornet protocols, in particular, the protocols presented in this dissertation.
Table B.3: The neighbor relation graph in the testbed depending on RF power. An edge between two vertices is assumed to exist in the graph if the wireless links between the nodes associated with these vertices meet certain requirements. For the RSSI metric, the RSSI value of the links in both directions has to be at least $-90$ dBm. For the LQI metric, the LQI value of the links in both directions has to be at least 95. Finally, for PLR metric, the packet loss rate of the links has to be below 15%.

Figure B.6: The testbed node degree distribution (RF power of $-25$ dBm). The figure depicts the distribution of the number of neighbors of a testbed node. In general, the node degree distribution is highly nonuniform, which is a common situation in the real world.
Figure B.7: The testbed path length distribution (RF power of $-25$ dBm). The figure illustrates the length distribution of the $55 \cdot 54 = 2970$ paths between all testbed nodes. A relatively large fraction of the paths has the same length as the network diameter, which is desirable when testing multihop routing protocols.
NUMEROUS pieces of software have been written to enable conducting the research presented in this dissertation. This appendix gives an overview of two major ones. First, it briefly surveys the hierarchical routing library, which was developed to conduct the experiments presented in Chapter 4. The library sources are publicly available. Second, it gives an overview of the library containing the four representative techniques from the whole state-stretch trade-off spectrum of point-to-point routing techniques, which was developed to conduct the experiments from Chapter 5. The library is ongoing work, but we are considering publishing its sources at some point.

C.1. HIERARCHICAL ROUTING LIBRARY

Parts of the hierarchical routing library used to conduct the experiments described in Chapter 4 have been made publicly available. More specifically, large parts of those library sources that implement hierarchical routing over a
landmark hierarchy were published. In addition, the published sources contain a sample application demonstrating the use of the library. The sources are available in the contributed code tree of the Concurrent Versions System (CVS) of TinyOS, a popular operating system for sensornets:

http://www.tinyos.net/

The sources can also be downloaded from my website:

http://www.few.vu.nl/~iwanicki/Ad_Hoc_Hierarchical_Routing/

The sources have the following structure:

**apps/** A directory containing the sample TinyOS application demonstrating the usage of the library.

**doc/** A directory containing documentation (currently empty).

**tools/** A directory with various tools for the library (currently empty).

**tos/** A directory with the library itself.

  **hierclust/** A directory that contains the code for hierarchical routing and the code for building and maintaining a landmark hierarchy.

  **interfaces/** A directory with common interfaces used throughout the library.

  **le/** A directory containing a passive beacon-based link estimator that can be used with the library.

  **sequencing/** A directory implementing sequencing of broadcast packets (necessary for link quality estimation).

  **utils/** A directory implementing various programming abstractions that are shared among the above components.

For more information, the reader should refer to the source code.

The sources can be compiled from the application directory. To this end, a TinyOS 2.0 distribution is required. Due to major refactoring of the interfaces in TinyOS 2.1, the sources do not compile with TinyOS 2.1. Adopting them for TinyOS 2.1 should be relatively easy, though. The resulting binaries can be installed on sensor nodes from the application directory. For more information on the compilation and installation process the reader should refer to the TinyOS tutorials, available on the above TinyOS website.
In addition to the TinyOS code with the library and the demo application, together with my brother, Krzysztof Iwanicki, we have developed visualization software for the demo application. The software is written in Java 5.0 and consists of proxies for connecting sensor nodes to the Internet, a control server for gathering statistics from and issuing commands for the nodes, and a client for visualizing traffic and controlling a network. The visualization software was used on a number of occasions to present live demos of hierarchical routing. A screenshot from the client application is depicted in Figure C.1.

![Figure C.1: A screenshot of the user interface of the visualization software. The software visualizes a sensor network and allows for issuing commands to the nodes, for instance, commands for starting routing flows. The software involves several layers (controlled by the top tool bar), such as connectivity layer, routing table layer, or routing address layer, which can be dynamically displayed or hidden. Moreover, it animates dynamically changing node state and packets flowing through the network.](image)

We are considering publishing the visualization software as well. If we decide to do so, the software will be available in the tools directory of the library.

---

1Krzysztof Iwanicki, M.Sc. student, Faculty of Mathematics and Information Science, Warsaw University of Technology, Pl. Politechniki 1, 00-661 Warszawa, Poland, e-mail: iwanicki.krzysztof@gmail.com.
C.2. GENERAL ROUTING LIBRARY

The general routing library, which was introduced in Chapter 5, is an ongoing project and is not yet publicly available for download. We are considering making its sources public at some time, though.

As of December 2009, the library comprises the four selected routing techniques (cf. Section 5.2) and has the following structure.

apps/ A directory containing the sample TinyOS application demonstrating the usage of the library.

docs/ A directory containing documentation, including minutes from our meetings and teleconferences regarding the library.

tos/ A directory with the library itself.

application/ A directory with various application-layer components, such as components implementing common statistics collection or components for controlling routing flows.

interfaces/ A directory containing the interfaces of the three routing abstractions.

le/ A directory with various implementations of the link quality estimation abstraction.

routing/ A directory implementing the routing state maintenance abstraction. Currently it also contains four subdirectories representing the modules grouping technique-specific hook functions.

transport/ A directory with various implementations of the packet forwarding abstraction.

types/ A directory with the type definitions in all routing abstractions.

uid/ A directory implementing unique node identification functionality.

utils/ A directory implementing various programming abstractions that are shared among all routing abstractions.

While in the final version the above directory structure may change, we do not expect large changes in the code.

The sources can be compiled from the sample application directory. Unlike the hierarchical routing library from the previous section, this library compiles with TinyOS 2.1. Likewise, the resulting binaries can be installed on sensor nodes from the application directory.
Appendix D

Foreign-Language Summaries

Since the formal regulations for a doctoral dissertation require an appendix with a Dutch summary of the dissertation, this appendix contains such a summary in Section D.1. The summary has been translated from English by Arno Bakker\textsuperscript{1}. As I am Polish, a Polish translation of the summary is also appended in Section D.2.

D.1. SAMENVATTING (DUTCH SUMMARY)

Hiërarchische Routering in Draadloze Netwerken met Weinig Vermogen

Draadloze sensornetwerken zijn een recente vorm van netwerken met weinig vermogen die een nieuwe klasse vormen binnen de informatica. Ze bestaan uit talrijke kleine draadloze apparaten die, ingebed in de fysieke ruimte, samen...

\textsuperscript{1}Arno Bakker, Scientific Programmer, Department of Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081A, 1081 HV Amsterdam, the Netherlands, e-mail: arno@cs.vu.nl.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission of the author and/or a fee.

verschillende kenmerken van de omgeving kunnen waarnemen en besturen. Hun doel is o.a. het uitbreiden van de digitale wereld van het Internet met de mogelijkheid om op afstand de fysieke ruimte waar te kunnen nemen en te controleren. Om dit doel te bereiken moeten zulke apparaatjes draadloos communiceren met elkaar en met andere apparatuur op het Internet.

De fundamentele functionaliteit die twee Internet apparaten in staat stelt om te communiceren is punt-tot-punt routering. Dit is de enige functionaliteit die geïmplementeerd is door alle Internet apparaten, zij het in de Internet kern of aan zijn randen. Aangezien een apparaat (een knoop) in het Internet alleen direct kan communiceren met een minieme deelverzameling van alle knopen (een netwerk kabel verbindt b.v. slechts twee knopen en een radio heeft slechts beperkt bereik) moet over het algemeen de data die door een knoop verzonden wordt door tussenliggende knopen doorgestuurd worden om de knoop van bestemming te bereiken (mogelijkerwijs aan de andere kant van de aardbol). Deze tussenliggende knopen vormen een pad en het doel van punt-tot-punt routering is het vinden van paden in een netwerk waarover data tussen knopen verzonden worden. Daarom is punt-tot-punt routering cruciaal in het Internet om communicatie mogelijk te maken tussen elk paar knopen, d.w.z., om de illusie van volledige connectiviteit te creëren in een netwerk waarin slechts weinig knopen direct kunnen communiceren.

In het begin werd punt-tot-punt routering niet noodzakelijk geacht voor sensornetwerken. Recent onderzoek heeft die visie veranderd: punt-tot-punt routering is belangrijk voor vele sensornetwerkapplicaties, in het bijzonder als sensorknopen volwaardige Internet apparaten moeten worden. Het doel van deze dissertatie is het ontwikkelen van een punt-tot-punt routeringsprotocol dat geschikt is voor sensornetwerken.

Het ontwikkelen van zo’n protocol is zeer uitdagend. Doordat sensorknopen ingebouwd zijn in de fysieke ruimte zijn ze sterk beperkt in termen van middelen, zoals energie, geheugen, bandbreedte en rekencapaciteit. Verder vertoont draadloze communicatie op laag vermogen verscheidene eigenaardige fenomenen die maken dat het onbetrouwbaar is en moeilijk te realiseren. Tenslotte worden de knopen vaak in grote aantallen toegepast om alle meet- en controlepunten af te dekker.

Behalve wat andere kleine uitdagingen is de grootste uitdaging die deze eigenschappen van sensornetwerken introduceren voor routering het simultaan garanderen van de volgende eigenschappen. Ten eerste, gegeven de beperkte middelen van een sensorknoop moet een routeringsprotocol slechts een kleine routing state hebben, wat noodzakelijk is voor schaalbaarheid en efficiëntie. Bovendien moet het protocol een kleine routing stretch bieden, d.w.z., de
routeringspaden die het vindt moeten dicht bij de optimale paden liggen, wat belangrijk is voor de efficiëntie en betrouwbaarheid. Dit reduceert namelijk de globale consumptie van middelen en verbetert de end-to-end datasyncelen. Verder moet het protocol bestand zijn tegen communicatiefouten en het falen van knopen, in het bijzonder gezien de onbetrouwbare aard van draadloze communicatie op laag vermogen en de interactie van de sensorknopen met de omgeving. Tenslotte moet het protocol zich in hoge mate zelf beheren, omdat dit de uitrol en het onderhoud van grote netwerken vereenvoudigt, welke inherent zijn in veel sensornetwerkapplicaties.

Deze vereisten worden in meer detail uitgelegd in Hoofdstuk 2, gebaseerd op een analyse van een aantal bestaande en een aantal toekomstige sensornetwerkapplicaties. Bovendien bevat deze analyse een sterk argument dat het onwaarschijnlijk is dat deze eisen op de korte termijn zullen veranderen door de voortgang van de technologie. Sensornetwerken zullen gebaseerd moeten zijn op laag vermogen, een kleine vormfactor, en lage prijs per eenheid. Dat betekent dat alle beperkingen en eigenaardigheden van sensornetwerken opgelost zullen moeten worden in software. Derhalve is het erg waarschijnlijk dat de onderzoeksresultaten die in deze dissertatie gepresenteerd worden voor lange tijd relevant zullen blijven.

Een analyse van gerelateerd werk over punt-tot-punt routering, zoals beschreven in het tweede deel van Hoofdstuk 2, levert enkele cruciale observaties op. Aan de ene kant bestaat er een heel spectrum van routeringstechnieken die potentieel geschikt zijn voor sensornetwerken, en die robuustheid en zelfbeheer beloven. Aan de andere kant toont de theorie over routering aan dat er een inherente afweging is tussen routing state en routing stretch die al deze technieken moeten maken. Kortgezegd: hoe kleiner de state in een techniek, hoe groter de stretch en vice versa. Daardoor kan het spectrum van technieken opgedeeld worden in een paar grote gebieden afhankelijk van de state-stretch afweging die zij maken. Vervolgens kan opgemerkt worden dat één van de meest veelbelovende gebieden in het techniekenspectrum, corresponderend met hiërarchische routering nog niet veel aandacht gekregen heeft. In het bijzonder is er naar mijn weten nog geen hiërarchisch routeringsprotocol voor sensornetwerken geïmplementeerd en geëvalueerd.

Het belangrijkste idee achter hiërarchische routering is om de knopen in een meerlaagse hiërarchie van clusters te organiseren. Met zo’n structuur kan een knoop, in plaats van routing state voor elke knoop in het netwerk bij te houden, slechts de routing state van een paar clusters in zijn nabijheid bijhouden. Op deze manier kan de routing state enorm gereduceerd worden tot een polylogaritmische functie van de grootte van de knopenpopulation.
Echter, zo’n reductie in routing state wordt afgewogen tegen een toename van routing stretch. In theorie kan de toename in stretch in hiërarchische routering groot zijn in bepaalde netwerktopologieën. Men kan speculeren dat dit de hoofdreden is waarom hiërarchische routering in sensornetwerken zo weinig onderzocht is.

Een andere plausibele reden is dat, zelfs als we de problemen veroorzaakt door beperkte middelen en de eigenaardigheden van sensornetwerken buiten beschouwing laten, hiërarchische routering vrij moeilijk te implementeren is. In het bijzonder het fundamentele probleem van deze routeringstechniek — het organiseren van de knopen in de clusterhiërarchie en het in stand houden van de hiërarchie wanneer knopen falen of de netwerkconnectiviteit verandert — is extreem complex. Deze twee kwesties maken hiërarchische routering onaantrekkelijk voor sensornetwerken, ondanks zijn andere merites.

Deze dissertatie betwist deze visie en promoot hiërarchische routering. Er zijn drie motiverende factoren achter hiërarchische routering. Ten eerste, hiërarchische routering biedt zeer kleine routing state. Ten tweede, hoewel de stretch in hiërarchische routering theoretisch groot kan zijn als arbitraire topologieën beschouwd worden, zijn de topologieën van sensornetwerken door hun inbedding in de fysische ruimte over het algemeen “geometrisch”. Dit betekent dat de lengte van een routeringspad snel groeit naarmate de knooppopulatie groeit. De theorie heeft bewezen dat in zulke topologieën hiërarchische routering zowel kleine state en kleine stretch kan bieden. Ten derde, het implementeren van hiërarchische routering hoeft niet noodzakelijkerwijs moeilijker te zijn dan het implementeren van andere technieken. Met name door het afzwakken van sommige van de hiërarchische eigenschappen kan men lokaal-werkende algoritmen ontwerpen waarin knopen zichzelf in een hiërarchie organiseren en deze autonoom onderhouden op een zeer robuuste en efficiënte manier. Derhalve, alles samen nemend, heeft hiërarchische routering de potentie om een overtuigende punt-tot-punt routeringstechniek voor sensornetwerken te zijn.

Om het bovenstaande argument te ondersteunen wordt in Hoofdstuk 3 een praktisch hiërarchisch routeringsprotocol voor sensornetwerken geïntroduceerd, genaamd PL-Gossip. Naar mijn weten is PL-Gossip het eerste hiërarchische routeringsprotocol dat effectief geïmplementeerd is voor sensornetwerken. Het ontleedt hiërarchische routering in drie algemeen erkende beheersbare abstracties die corresponderen met verschillende aspecten van routering in sensornetwerken, te weten het schatten van de verbindingskwaliteit, het onderhoud van de routing state en het doorsturen van netwerkpakketen. Het legt uit hoe elk van deze abstracties geïmplementeerd kan worden om hiërarchische routering te
ondersteunen.

De meest uitdagende en cruciale routeringsabstractie in hiërarchische routering is onderhoud van de routing state, d.w.z., het onderhoud van de clusterhiërarchie zichtbaar in de routingstabellen en routeringsadressen van knopen. Zoals eerder opgemerkt is de complexiteit van dit probleem waarschijnlijk één van de redenen dat hiërarchische routering nog niet geïmplementeerd en geëvalueerd is in sensornetwerken. Daarom legt PL-Gossip extra nadruk op het probleem van het onderhouden van de clusterhiërarchie.

Om het probleem praktisch te hanteren te maken richt PL-Gossip zich op best-effort hiërarchieën in plaats van op optimale hiërarchieën. Met name laat Hoofdstuk 3 zien hoe de eigenschappen van een clusterhiërarchie aangepast kunnen worden zodat bewezen (zie Appendix A) kan worden dat de hiërarchie: ten eerste, geschikt is voor hiërarchische routering, ten tweede, het potentieel heeft om kleine routing state te bieden, en ten derde, onderhouden kan worden door echte sensorknopen. Zulke aanpassingen hoeven echter niet altijd dezelfde te zijn als die in Hoofdstuk 3. Ze worden namelijk veranderd in daaropvolgende hoofdstukken van de dissertatie, zelfs om verschillende typen hiërarchieën te beschrijven. Dit is slechts één van de voorbeelden waaruit blijkt dat PL-Gossip ideeën introduceert die breder toepasbaar zijn.

Om de eigenschappen van een clusterhiërarchie te onderhouden stelt de variant op PL-Gossip gepresenteerd in Hoofdstuk 3 voor om een combinatie van twee simpele concepten te gebruiken: lokale operaties voor het wijzigen van de hiërarchie en asynchroon lokaal roddelen (gossip) voor het verspreiden van zulke modificaties naar de betreffende knopen. Dit is voldoende om de knopen die PL-Gossip draaien zichzelf te laten organiseren in een clusterhiërarchie, beschreven door een verzameling specifieke eigenschappen (en deze te onderhouden). Echter, al zijn deze twee concepten simpel, het combineren ervan om autonoom onderhoud van de hiërarchie mogelijk te maken creëert een aantal uitdagingen. Zulke uitdagingen zijn verbonden aan, b.v. het consistent toepassen van hiërarchiemodificaties door de betreffende knopen, of het kiezen van clusters op laag niveau die gepromoveerd zullen worden naar een hoger niveau. Veel inspanning is gericht op het bieden van oplossingen voor deze niet-triviale uitdagingen en om de correctheid van de oplossing analytisch te bewijzen. Deze oplossingen zijn echter simpel waardoor de implementatie van PL-Gossip op echte sensorknopen gefaciliteerd wordt. Alles bij elkaar illustreert Hoofdstuk 3 dat men inderdaad een praktisch autonoom-beheerd hiërarchisch routeringsprotocol voor sensornetwerken kan ontwerpen.

De ontwikkeling van PL-Gossip maakt het mogelijk om hiërarchische routering experimenteel te evalueren in verscheidene opstellingen van
APPENDIX D. FOREIGN-LANGUAGE SUMMARIES

sensornetwerken, wat het onderwerp is van Hoofdstuk 4. De evaluatie gebruikt drie experimentele platformen: een hoog-niveau simulator; TOSSIM, een laag-niveau simulator voor sensornetwerken met realistische modellen van draadloze communicatie op laag vermogen; en een echt testbed van meer dan 50 knopen dat ik aan de Vrije Universiteit gebouwd heb (zie Appendix B). Hiërarchische routering wordt geëvalueerd met betrekking tot de eerder genoemde vereisten voor een routeringsprotocol voor sensornetwerken: routing state, routing stretch en robuustheid.

De evaluatie laat zien dat hoewel het niet gericht is op optimale clusterhiërarchieën het inderdaad zeer kleine routing state kan bieden. Bovendien kan PL-Gossip ondanks zo’n kleine state in de praktijk ook kleine stretch bieden, gemiddeld binnen 50% van de optimale waarde, wat het gevolg is van de eerdergenoemde geometrische aard van sensornetwerken. Tenslotte is PL-Gossip ook robuust in de zin dat fouten relatief weinig verstoring veroorzaken en dat het protocol hiervan relatief snel herstelt. Deze resultaten suggereren dat hiërarchische routering inderdaad een aantrekkelijke punt-tot-punt routeringstechniek is voor sensornetwerken.

Verder beperkt de evaluatie van hiërarchische routering in Hoofdstuk 4 zich niet tot het beschreven PL-Gossip protocol. Zoals boven beargumenteerd is PL-Gossip geen vastgelegd en monolitisch protocol maar vormt het een basis voor een protocol en introduceert het ideeën die breder toepasbaar zijn en waarmee geëxperimenteerd kan worden. Daarom is er i.p.v. alleen een PL-Gossip implementatie een volledige hiërarchische routeringsbibliotheek geïmplementeerd voor de evaluatie. De bibliotheek definiert een gemeenschappelijk raamwerk voor een hiërarchisch routeringsprotocol voor sensornetwerken, gebaseerd op de lessen die geleerd zijn uit PL-Gossip. Tegelijkertijd echter identificeert het een aantal ontwerppunten. Door de oplossingen op deze ontwerppunten te variëren kan men verschillende mechanismen (en nieuwe ideeën) voor hiërarchische routering evalueren en vergelijken. De broncode van de bibliotheek is publiek beschikbaar gemaakt (zie Appendix C.1).

De bibliotheek maakt het uitvoeren van diepgaande evaluaties van verschillende ontwerpbeslissingen mogelijk die van invloed zijn op routing state, routing stretch en de robuustheid van hiërarchische routering zelf. Eén van zulke ontwerppunten die bestudeerd wordt in Hoofdstuk 4 is het type en de eigenschappen van een cluster hiërarchie. In het bijzonder wordt aangetoond dat een landmark-hiërarchie normaliter grotere state vereist dan een area-hiërarchie, maar daar voor in de plaats biedt het kleinere stretch, gemiddeld ongeveer binnen 25% van de optimale waarde. Deze kan verder gewijzigd worden door
specifieke eigenschappen van de hiërarchie te variëren. Met andere woorden, door het type of de eigenschappen van een clusterhiërarchie te variëren kan men de afweging tussen state en stretch in hiërarchische routering zelf verder exploreren. Een ander ontwerp punt dat bestudeerd wordt in Hoofdstuk 4 is de methode voor het propageren van informatie over de hiërarchie. Behalve lokaal asynchroon roddelen zoals gebruikt door de variant van PL-Gossip in Hoofdstuk 3 worden een bekende op flooding gebaseerde methode en een nieuwe hybride methode bestudeerd. Kortom, er wordt aangetoond dat verschillende methoden verschillende aspecten van robuustheid beïnvloeden, dus wederom: hiërarchische routering kan geoptimaliseerd worden naar een specifieke gekozen metriek. Alles bij elkaar bevestigt de evaluatie van hiërarchische routering uitgevoerd in Hoofdstuk 4 dat hiërarchische routering aantrekkelijk is voor sensornetwerken, maar laat ook zien dat het aangepast kan worden voor specifieke toepassingen.

Aangezien het onderzoek in Hoofdstuk 3 en Hoofdstuk 4 het bovengenoemde gat vult in het spectrum van routeringstechnieken maakt dit het mogelijk om representatieve technieken uit het gehele spectrum experimenteel te vergelijken. Zo’n initiële vergelijking is het onderwerp van Hoofdstuk 5.

Om deze vergelijking uit te voeren is er een andere bibliotheek geïmplementeerd, die deze keer verschillende technieken omvat, zoals beschreven in Hoofdstuk 5. De bibliotheek omvat vier van deze technieken, die samen het hele spectrum van state-stretch afwegingen representeren: shortest-path routering, welke de grootste state vereist maar minimale stretch biedt, compacte en hiërarchische routering, die met verschillende granulariteit state reduceren ten koste van stretch, en constant-state routering, welke de kleinste state nodig heeft maar de grootste stretch oplevert. Het grootste deel van de code wordt gedeeld door alle technieken, wat de implementatie van deze technieken voor een groot deel uniform maakt. De bibliotheek is geëvalueerd in TOSSIM en met de twee testbeds: een testbed van 50+ knopen en van 100+ knopen. De evaluatie richt zich op de afweging state/stretch.

Naast wat kleine resultaten levert de evaluatie twee grote bijdragen. Ten eerste, illustreert een algemene eigenschap van sublinear-state routeringstechnieken. Specifieker gezegd, door de geometrische aard van sensornetwerkinstallaties bieden de meeste technieken met sublinear-state een routing stretch welke dicht bij de optimale ligt. Met andere woorden, in sensornetwerken houdt een grote reductie in routing state normaliter slechts een kleine toename van routing stretch in. Dit leidt tot de tweede bijdrage, welke direct gerelateerd is aan hiërarchische routering. Specifieker gezegd, hiërarchische routering biedt een grote reductie in state met slechts een kleine
toename in stretch in vergelijking met de andere representatieve technieken. In het bijzonder biedt hiërarchische routering, in vergelijking met compacte routering, een state die ten minste een orde van grootte betere schaalbaarheid toestaat, terwijl de stretch maar weinig beïnvloed wordt met minder dan 10–15% gemiddeld, wat voor sensornetwerktotopassingen de prestaties niet significant zal verslechteren. Alles bij elkaar suggereert Hoofdstuk 5 (behalve dat het ook het argument dat hiërarchische routering een aantrekkelijke techniek voor sensornetwerken is versterkt) dat vanuit het perspectief van het spectrum van state/stretch afwegingen, hiërarchische routering een afweging biedt die aantoonbaar het meest aantrekkelijk is voor vele sensornetwerktotopassingen.

Samenvattend, de conclusie die getrokken kan worden uit alle onderzoekresultaten is dat de these zoals geformuleerd in deze dissertatie geldig is.

Hiërarchische routering is een overtuigende punt-tot-punt routeringstechniek voor grote sensornetwerken. In de praktijk biedt het niet alleen kleine routing state, maar ook kleine routing stretch. Bovendien is het mogelijk om robuuste, efficiënte, autonome hiërarchische routeringsprotocollen aan te bieden die in de praktijk werken.

D.2. STRESZCZENIE (POLISH SUMMARY)

Trasowanie hierarchiczne w sieciach bezprzewodowych
o niskim poborze mocy

Bezprzewodowe sieci sensorowe (z ang. wireless sensor networks, w skrócie sensorsnets lub WSNs) to nowatorskie sieci o niemal zerowym poborze mocy stanowiące nową klasę urządzeń komputerowych. Składają się one z wielu bezprzewodowych mini-komputerków, które mogą być wbudowane w fizycznie otaczające nas obiekty po to by wspólnie obserwować i kontrolować różne aspekty naszego otoczenia. Ich głównym zadaniem jest więc wzbogacenie cyfrowego świata Internetu możliwościami zdalnej obserwacji i bezpośredniego wpływania na otaczający nas świat fizyczny. Aby spełniać to zadanie, te mini-komputerki (zwane również sensorkami) muszą mieć możliwość bezprzewodowej komunikacji ze sobą oraz z innymi urządzeniami podłączonymi do Internetu.2

2Jako że niniejsza praca zawiera prawdopodobnie jedno z pierwszych tłumaczeń zwrotu “wireless sensor networks” na język polski, pozwoliłem sobie przetłumaczyć ten zwrot jako
Fundamentalnym mechanizmem umożliwiającym komunikację pomiędzy dwoma dowolnymi urządzeniami w Internecie jest trasowanie punkt-do-punktu (z ang. point-to-point routing). Jest to jedyny mechanizm, który musi być zaimplementowany przez wszystkie urządzenia tworzące Internet: zarówno przez urządzenia indywidualnych użytkowników, jak również przez urządzenia będące częścią globalnej infrastruktury. Każde urządzenie w Internecie (tzw. węzeł) może komunikować się bezpośrednio jedynie z niewielką liczbą innych węzłów, przykładowo kabel sieciowy łączy tylko dwa węzły, radio zaś ma ograniczony zasięg. Dlatego też dane wysłane przez jakiś węzeł źródłowy do jakiegoś innego węzła docelowego, potencjalnie w innym punkcie kuli ziemskiej, muszą być przekazywane przez kolejne węzły pośredniczące, aby mogły ostatecznie dotrzeć do celu. Takie węzły pośredniczące tworzą tak zwaną trasę. Trasowanie punkt-do-punktu ma zatem za zadanie znajdowanie tras dla danych przesyłanych w sieci i w ten sposób umożliwiać komunikację pomiędzy wszystkimi jej węzłami. Z perspektywy Internetu trasowanie punkt-do-punktu jest więc kluczowym mechanizmem dla aplikacji, ponieważ w sieci z niewielką liczbą faktycznych połączeń miedzy węzłami tworzy ono iluzję, iż każdy węzeł jest bezpośrednio połączony z każdym inny.


Opracowanie takiego protokołu jest jednak niezwykle trudne. Z powodu konieczności długotrwałej, praktycznie nienadzorowanej pracy wewnątrz, na powierzchni lub w pobliżu otaczających nas obiektów fizycznych sensorki mają wybitnie ograniczone zasoby, takie jak energia, pamięć operacyjna, przepustowość kanału komunikacyjnego i moc obliczeniowa. Ponadto używana przez sensorki bezprzewodowa komunikacja radiowa o ultra-niskim poborze mocy jest podatna na wiele zjawisk fizycznych, które sprawiają, że w efekcie transmitowane dane mogą permanentnie lub tylko przez pewien czas nie docierać do odbiorcy, nawet jeśli wydaje on się być w zasięgu radia nadawcy. Do tego wszystkiego, sieci sensorowe składają się zwykle z dużej liczby mini-urządzeń po to by pokryć wszystkie punkty wymagające obserwacji i wszystkie kontrolowane

“bezprzewodowe sieci sensorowe” zamiast bardziej dosłownie jako “bezprzewodowe sieci czujników”. Według mnie takie tłumaczenie lepiej odzwierciedla fakt, iż sensorki w sieci obserwują otaczający nas świat współpracując ze sobą.

Powyższe wymagania stawiane protokołom trasującym i ich geneza wy tłumaczone są dokładnie w Rozdziale [2] na podstawie analizy istniejących i przewidywanych zastosowań bezprzewodowych sieci senzorowych. Analiza ta stanowi również mocny dowód na to, że jest mało prawdopodobne, aby te wymagania zmieniły się w niedalekiej przyszłości. Unikając zbytniego wdawania się w szczegóły, jest to spowodowane faktem, iż aby sieci sensorowe mogły być masowo wykorzystywane w codziennym życiu, postęp w technologii sprzętu używanego do budowy sensorów musi dalej skupiać się na redukcji poboru mocy, wielkości i ceny pojedynczego sensora, tym samym wymagając, aby to oprogramowanie a nie sprzęt radziło sobie z ograniczeniami zasobów sensorów i wadami komunikacji radiowej, której używa. W efekcie jest wiele prawdopodobne, że wyniki badań zaprezentowane w niniejszej pracy pozostaną aktualne przez dłuższy okres czasu.
Analiza literatury istniejącej na temat trasowania dokonana w drugiej części Rozdziału 2 dostarcza kilku kluczowych obserwacji. Z jednej strony istnieje całe spektrum różnych technik trasowania, które, oferując odporność na błędy i awarie oraz zdolności samozarządzania, mogą być potencjalnie używane w bezprzewodowych sieciach sensorowych. Z drugiej strony teoria trasowania dowodzi, że każdej technice towarzyszy nieodłączny kompromis pomiędzy wielkością informacji trasującej a rozciąganiem trasy — w skrócie — im mniej informacji trasującej dana technika wymaga tym bardziej rozciągnięte trasy ta technika oferuje i odwrotnie. Spektrum technik trasowania może więc być podzielone na kilka głównych obszarów w zależności od konkretnego kompromisu pomiędzy wielkością informacji trasującej a rozciąganiem trasy, jaki towarzyszy poszczególnym technikom. Można wtedy zaobserwować, że na temat jednego z najbardziej obiecujących obszarów — obszar u odpowiadającemu trasowaniu hierarchicznemu (z ang. *hierarchical routing*) — zrobiono niewiele praktycznych badań. W szczególności w przeciwieństwie do pozostałych technik trasowania, zgodnie z moją najlepszą wiedzą, żaden protokół wykorzystujący trasowanie hierarchiczne nie został dotychczas zaimplementowany i przetestowany w bezprzewodowych sieciach sensorowych.

Podstawowym pomysłem wykorzystywanym w trasowaniu hierarchicznym jest zorganizowanie wszystkich węzłów w wielopoziomową wirtualną hierarchię grup, tak zwanych klastrów. Dzięki tej organizacji każdy węzeł nie musi przechowywać informacji trasującej dla każdego innego węzła w sieci, lecz jedynie informacje na temat kilku nieodległych klastrów na kolejnych poziomach hierarchii. W ten sposób wielkość informacji trasującej może być znacznie zredukowana z liniowej do polilogarytmicznej funkcji liczby węzłów w sieci.

Jednakże zgodnie z wyżej wspomnianym kompromisem taka redukcja wielkości informacji trasującej prowadzi do zwiększenia rozciągania znajdowanych tras. W teorii rozciąganie trasy w trasowaniu hierarchicznym może być znaczne w niektórych topologii sieciowych. Można spekulować, że to potencjalnie duże rozciąganie trasy jest jednym z powodów niewielkiej ilości badań dotyczących trasowania hierarchicznego w bezprzewodowych sieciach sensorowych.

Kolejnym możliwym powodem jest fakt, iż zaimplementowanie trasowania hierarchicznego jest samo w sobie dość trudne, nawet pomijając trudności wynikające z niezmiernie ograniczonych zasobów sensorów i z komunikacji radiowej, która wykorzystują. Zwłaszcza fundamentalny problem tej techniki trasującej — zorganizowanie węzłów w wirtualną hierarchię klastrów i
utrzymywanie tej hierarchi, gdy wezły ulegają awariom a łączność między nimi ulega zmianom — jest szczególnie złożony. Te dwie powyższe kwestie mogą sprawiać, że pomimo wielu zalet trasowanie hierarchiczne jest odbierane jako nieatrakcyjne dla bezprzewodowych sieci sensorowych.

Niniejsza praca kwestionuje ten punkt widzenia i promuje trasowanie hierarchiczne. Istnieją trzy główne czynniki przemawiające za tę technikę. Po pierwsze, ogromna zaleta trasowania hierarchicznego jest to, iż wymaga ono niezwykle małej ilości informacji trasującej. Po drugie, mimo że teoretycznie rozciąganie trasy w trasowaniu hierarchicznym może być duże jeśli rozważymy dowolne topologie sieciowe, z powodu wbudowania w otaczający nas świat topologie bezprzewodowych sieci sensorowych są zwykle “geometryczne”, co oznacza, że długość trasy rośnie w nich szybko wraz ze wzrostem liczby węzłów w sieci. Teoria dowodzi, iż w takich topologiach trasowanie hierarchiczne może równocześnie oferować i mała ilość informacji trasującej, i małe rozciąganie tras. Po trzecie, zaimplementowanie trasowania hierarchicznego nie musi być koniecznie trudniejsze niż zaimplementowanie innych technik trasowania. W szczególności, rozluźniając niektóre właściwości hierarchii klastrów można stworzyć zdecentralizowane algorytmy, w których wezły samo-organizują się w taka hierarchie i samodzielnie utrzymują ją w sposób wydajny i odporny na rozmaite awarie i błędy. Biorąc pod uwagę te trzy czynniki, trasowanie hierarchiczne ma potencjał niezbędny, aby być idealną techniką dla bezprzewodowych sieci sensorowych.


Najbardziej skomplikowaną a zarazem kluczową abstrakcją w trasowaniu hierarchicznym jest utrzymywanie informacji trasującej, to jest, konstrukcja i utrzymywanie wyżej wspomnianej hierarchii klastrów, której różne fragmenty są odzwierciedlone w lokalnej informacji trasującej każdego wezła. Jak również zostało wspomniane wyżej, złożoność tego problemu jest jednym z prawdopodobnych powodów, dla których trasowanie hierarchiczne nie
zostało dotychczas zaimplementowane i przetestowane na sieciach sensorowych. Dlatego też ogromna większość opisu protokołu PL-Gossip dotyczy właśnie konstrukcji i utrzymywania hierarchii klastrów.

W celu sprawienia, aby problem konstrukcji i utrzymywania hierarchii klastrów stał się praktycznie rozwiązywalny, PL-Gossip zakłada, iż hierarchie, które będzie budował i utrzymywał nie muszą koniecznie być optymalne, lecz mogą być dostosowane w zależności od potrzeb. W szczególności, Rozdział 2 ilustruje jak można dostosować właściwości hierarchii klastrów tak, aby można było udowodnić (zob. Dodatek A), że taka hierarchia: po pierwsze, nadaje się do trasowania hierarchicznego, po drugie, może wymagać niewiele informacji trasującej i, po trzecie, może być zbudowana i utrzymywana przez prawdziwe sensorki bez pomocy ludzkiego administratora. Te właściwości można jednak zmieniać, co jest de facto robione w kolejnych rozdziałach niniejszej pracy, nawet do punktu, w którym opisują one zupełnie inne klasy hierarchii. Jest to tylko jeden z przykładów pokazujących, że PL-Gossip nie jest sztywnym, monolitycznym protokołem, lecz pomysły, które przedstawia, mają znacznie szersze zastosowanie.

Do stworzenia mechanizmów umożliwiających budowę i utrzymywanie hierarchii spełniającej wybrany zestaw właściwości, wariant protokołu PL-Gossip opisany w Rozdziale 3, proponuje połączyć dwie nieskomplikowane idee: lokalne operacje, pozwalające indywidualnym węzłom modyfikować hierarchię klastrów, oraz lokalne asynchroniczne plotkowanie (z ang. gossiping), pozwalające propagować takie lokalne modyfikacje hierarchii do innych węzłów. Pomimo że te dwie idee są dość nieskomplikowane, połączenie ich tak, aby umożliwić węzłom samodzielne zorganizowanie się w hierarchię klastrów spełniającą jakiś wybrany zestaw właściwości a następnie dalsze utrzymywanie tej hierarchii, gdy łączność pomiędzy węzłami ulega zmianom lub same węzły ulegają awariom, prowadzi do wielu problemów. Przykłady takich problemów to spójne adoptowanie lokalnie wykonanych modyfikacji hierarchii przez wszystkie węzły, których takie modyfikacje dotyczą, czy też efektywna elekcja tych klastrów na niższych poziomach hierarchii, które staną się załączkiem klastrów na wyższych poziomach. Dużo wysiłku zostało włożone, aby dostarczyć rozwiązań dla tych nietrywialnych problemów i aby udowodnić poprawność tych rozwiązań analitycznie. Mimo wszystko jednak, podobnie jak dwie wyżej wymienione idee, rozwiązania pozwalające te idee połączyć w spójny protokół są dość proste, co ułatwia implementację protokołu PL-Gossip na prawdziwe sensorki. Ogólnym wnioskiem płynącym z Rozdziału 3 jest obserwacja, iż opracowanie praktycznego, samozarządzającego się protokołu trasowania hierarchicznego dla sieci sensorowych jest faktycznie możliwe.
Oprócz tego protokół PL-Gossip umożliwia przeprowadzanie eksperymentalnych badań nad trasowaniem hierarchicznym w bezprzewodowych sieciach sensorowych. Takie pionierskie badania są tematem Rozdziału 4. Zostały one przeprowadzone na trzech eksperymentalnych platformach: w specjalnym wysokopoziomowym symulatorze sieci sensorowych, który samodzielnie napisałem, w symulatorze TOSSIM, to jest, niskopoziomowym symulatorze sensorów zawierającym realistyczne modele bezprzewodowej komunikacji o niskim poborze mocy, oraz w rzeczywistej bezprzewodowej sieci sensorowej składającej się z 50+ węzłów, którą zbudowałem na swoim uniwersytecie (zob. Dodatek B). Trasowanie hierarchiczne jest oceniane z punktu widzenia wyżej wymienionych właściwości kluczowych dla protokołów trasowania w sieciach sensorowych, to jest, ilości informacji trasującej, rozciągnięcia tras i odporności na błędy komunikacji i awarie sensorów.

 Wyniki przeprowadzonych badań pokazują, iż pomimo rezygnacji z konstrukcji i utrzymywania optymalnych hierarchii klastrów, protokół PL-Gossip faktycznie wymaga niewielkiej ilości informacji trasującej. Niemniej jednak pomimo takiej malej ilości informacji trasującej PL-Gossip oferuje również dość małe rozciąganie trasy, średnio poniżej 50% w stosunku do optymalnej trasy. Jest to konsekwencja wyżej wspomnianych geometrycznych właściwości bezprzewodowych sieci sensorowych. Ponadto PL-Gossip jest dość odporny na różnego rodzaju błędy i awarie — takie zdarzenia powodują zwykle małe zakłócenia w pracy protokołu a ponadto protokół szybko je wykrywa i wykonuje niezbędne czynności naprawcze. W sumie wyniki te sugerują, że trasowanie hierarchiczne jest faktycznie atrakcyjną techniką trasowania punkt-do-punktu dla bezprzewodowych sieci sensorowych.

D.2. STRESZCZENIE (Polish Summary)

(zob. Dodatek C.1).


Badania zaprezentowane w Rozdziałach 3. i 4. wypełniają wcześniej wspomnianą lukę w badaniach dotyczących spektrum technik trasowania dla bezprzewodowych sieci sensorowych. Tym samym dzięki nim można wreszcie eksperymentalnie porównać reprezentatywne techniki z całego spektrum. Takie wstępne porównanie jest tematem Rozdziału 5.

W celu przeprowadzenia tego porównania została opracowana kolejna biblioteka trasowania dla sieci sensorowych, tym razem obejmująca różne techniki trasowania. Dokładniej, biblioteka ta obejmuje cztery techniki, które z perspektywy kompromisu pomiędzy ilością informacji trasującej a rozciągnięciem tras razem reprezentują całe spektrum: trasowanie wzdłuż najkrótszych ścieżek (z ang. *shortest-path routing*), które wymaga największej ilości informacji trasującej, ale oferuje najmniejsze rozciągnięcie tras, trasowanie
kompaktowe (z ang. compact routing) i trasowanie hierarchiczne (z ang. hierarchical routing), które w różnym stopniu redukują ilość informacji trasującej kosztem zwiększenia rozciągnięcia tras, oraz trasowanie ze stałą ilością informacji trasującej niezależnie od wielkości sieci (z ang. constant-state routing), które wymaga najmniejszej ilości informacji trasującej, ale jednocześnie charakteryzuje się największym rozciągnięciem tras. Kod źródłowy biblioteki jest w ogromnej większości wspólny dla wszystkich czterech technik, co w znacznym stopniu ujednolica ich implementacje. Implementacje tego pełnego spektrum technik zostały przetestowane w wyżej wspomnianym symulatorze TOSSIM i w dwóch rzeczywistych bezprzewodowych sieciach sensorowych składających się odpowiednio z 50+ oraz ze 100+ węzłów. Przeprowadzone eksperymenty miały głównie na celu zbadanie kompromisu pomiędzy ilością informacji trasującej a rozciągnięciem tras, jaki każda z technik oferuje w praktyce.

Pomijając mniej istotne wyniki, badania te pozwalają wyciągnąć dwa główne wnioski. Po pierwsze, ilustrują one ogólną własność technik trasowania z podliniową ilością informacji trasującej, to jest technik, w których żaden węzeł nie przechowuje informacji trasującej dla wszystkich pozostałych węzłów w sieci. Z powodu wyżej wspomnianych geometrycznych właściwości bezprzewodowych sieci sensorowych techniki takie zwykle oferują trasy, które są niewiele rozciągnięte w porównaniu do optymalnych tras. Innymi słowy, w sieciach sensorowych, duża redukcja ilości informacji trasującej zwykle powoduje tylko niewielkie zwiększenie rozciągnięcia tras. To automatycznie prowadzi do kolejnego wniosku, który jest bezpośrednio związany z trasowaniem hierarchicznym. W porównaniu z pozostałymi technikami, trasowanie hierarchiczne oferuje duża redukcja informacji trasującej przy jednoczesnym niewielkim wzroście rozciągnięcia tras. W szczególności, w porównaniu z trasowaniem kompaktowym, ilość informacji trasującej w trasowaniu hierarchicznym jest na tyle mniejsza, że sieci używające trasowania hierarchicznego mogą być o rząd wielkości większe od sieci używających trasowania kompaktowego. Jednocześnie trasowanie hierarchiczne oferuje trasy, które są rozciągnięte średnio o jedynie 10–15% więcej niż trasy oferowane przez trasowanie kompaktowe, co ma bardzo niewielki negatywny efekt na wydajność aplikacji. Ogólnie, wyniki zaprezentowane w Rozdziale [3] nie tylko dostarczają dodatkowy mocny argument na to, że trasowanie hierarchiczne może być atrakcyjne dla bezprzewodowych sieci sensorowych, ale również sugerują, że spośród reprezentacyjnych technik z całego spektrum technik trasowania, kompromis pomiędzy ilością informacji trasującej a rozciągnięciem tras oferowany przez trasowanie hierarchiczne jest zapewne najbardziej pożądany.
w wielu zastosowaniach sieci sensorowych.

Podsumowując, ogólnym wnioskiem płynącym z całości wyżej przedstawionych badań jest fakt, iż teza sformułowana i broniona w niniejszej pracy wydaje się być prawdziwa: