ANYPATH ROUTING

THÈSE N^{O} 3636 (2006)

PRÉSENTÉE LE 10 NOVEMBRE 2006
À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
Laboratoire de communications audiovisuelles 1
SECTION DES SYSTÈMES DE COMMUNICATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Henri DUBOIS-FERRIERE

ingénieur en systèmes de communication diplômé EPF
de nationalité suisse et originaire de Meyrin (GE)

acceptée sur proposition du jury:

Prof. E. Telatar, président du jury
Prof. M. Vetterli, Prof. M. Grossglauser, directeurs de thèse
Prof. D. Estrin, rapporteur
Prof. T. Gross, rapporteur
Dr S. McCanne, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Lausanne, EPFL
2006

Abstract

In many networks, it is less costly to transmit a packet to *any* node in a set of neighbors than to one specific neighbor. A well-known instance is with unreliable wireless links, where the probability that at least one node out of n receives a packet increases with n. This observation was previously exploited, by modifying single-path routing to assign to each node group of *candidate* next-hops for a particular destination. However, single-path metrics not do reflect the cost of forwarding when a sender has multiple candidate relays, and they result in routing decisions that are in many cases suboptimal.

This dissertation addresses the *shortest anypath routing* problem: how to assign a set of candidate relays at each node for a given destination, such that the cost of forwarding a packet to the destination is minimized. The key is the following tradeoff: on the one hand, increasing the number of candidate relays decreases the forwarding cost, but on the other, this increases the likelihood of ”veering” away from the most direct trajectory.

Solving the shortest anypath problem requires us first to formalize the notions of anycast link cost and remaining path cost, that generalize the link cost and path cost of single-path routing. Unlike with single-path routing, a packet can travel across an anypath route in many different ways; the cost of this route is naturally defined as the expected cost of all possible traversals. We introduce an algorithm that provably computes the shortest anypath route between each node and a destination. It is based on the Bellman-Ford algorithm and so is amenable to implementation in a distributed setting. This algorithm works for all *physical* cost metrics; we show that there exist certain “non-physical” cost metrics under which the shortest anypath route may contain cycles. We also explore the interaction between the relay selection policy, that is, the way in which the effective relay is chosen among many receivers, and the cost of optimal routes. We also explore the robustness of anypath routes, and find that they are significantly more stable in the face of topology changes and imperfect information than are single-path and multipath routes.

The principles of anypath routing are general and can be applied in many settings. Our application focus in this dissertation is on low-power, low-rate wireless communication in embedded wireless networks. We introduce novel ways in which low-power link-layers can take advantage of anycast forwarding to reduce transmission energy or latency. We describe the design and implementation of a complete anypath routing protocol; evaluation on a 50-node wireless testbed demonstrates that anypath routing is robust, stable, and increases energy efficiency of low-power nodes by a significant factor over the equivalent system using single-path routing.

Finally, we describe a novel error-correction mechanism for multi-hop wireless communication based on packet combining. Packet combining allows to exploit corrupt packets received at different nodes by jointly decoding independent, noisy versions of an original packet.

Keywords: Wireless networks, routing, diversity

Résumé

Dans un réseau de communication, il est souvent moins coûteux de transmettre un paquet à n'importe quel noeud parmi un ensemble de voisins qu'à un voisin spécifique. Un exemple bien connu survient avec des liens non fiables, ou la probabilité qu'un noeud parmi n recoive le paquet augmente avec n. Cette observation fut précédemment exploitée, en modifiant le routage à chemin unique afin d'assigner à chaque noeud un ensemble de relais *candidats* pour une destination donnée. Cependant, le coût d'un chemin unique ne reflète pas le coût de transmission avec plusieurs candidats, et par conséquent le choix de routes résultant n'est souvent pas optimal.

Cette thèse a pour sujet le problème du *plus court routage anypath*. Il s'agit de désigner les relais candidats à chaque noeud de maniere à ce que le coût total de transmission d'un paquet jusqu'à sa destination soit minimisé. La clef est d'équilibrer au mieux la tension suivante: d'un côté, le coût de transmission décroît lorsque l'on augmente le nombre de relais candidat; de l'autre la probabilité de s'écarter de la trajectoire directe est accrue.

La résolution de ce problème nécessite en premier lieu de formaliser la notion de coût de transmission *anycast*, et de coût de chemin restant. Contrairement aux routes à chemin unique, les routes anypath peuvent acheminer un paquet par plusieurs chemins différents; le coût d'une route anypath est donc naturellement défini comme le coût moyen de tous ses chemins. Nous introduisons un algorithme qui calcule les routes anypath les plus courtes entre chaque source et une destination. L'algorithme est basé sur celui de Bellman-Ford et peut aisément être implementé dans un context distribué. Il fonctionne pour tout modèle de coût *physique*; nous montrons qu'il existe certains modèles de coûts non-physiques avec lesquels la plus courte route anypath peut contenir des cycles. Nous explorons aussi la robustesse des routes anypath, et nous trouvons qu'elles sont plus stables face aux changements de topologie que ne le sont les routes à chemin unique.

Les principes du routage anypath sont généraux et peuvent s'appliquer à de nombreux contextes. Cette thèse à pour domaine d'application principal la communication à basse consommation et à faibles débits dans les réseaux sans-fil embarqués. Nous introduisons de nouveaux méchanismes de transmission anycast afin de réduire soit le délai, soit le coût énergétique de transmission. Nous décrivons le design et l'implémentation d'un protocole de routage anypath complet; une évaluation sur un lit de test montre que le routage anypath est robuste, stable, et diminue la consommation énergetique de noeuds à basse puissance d'un facteur significatif.

Enfin, nous décrivons un nouveau méchanisme de correction d'erreurs pour la communication sans fil à sauts multiples, basé sur la combinaison de paquets. Ce méchanisme permet de tirer avantage de paques corrompus recus par différents noeuds en décodant conjointement plusieurs copies indépendemment bruitées d'un paquet.

Mots-clés: Réseaux sans fil, routage, diversité

Acknowledgements

I would like to express my sincere gratitude to my two thesis advisors, Martin Vetterli and Matthias Grossglauser. I have known Martin since my undergraduate time at EPFL, and the opportunity to work with him was a large factor in my decision to return to EPFL and enter the PhD program after working two years in San Francisco. Both Martin and Matthias have been model advisors, combining insight, support, enthusiasm, and a "no-strings attached" form of guidance that always left me the freedom and independence to choose the research avenues that I wished to pursue. Just as importantly, meetings with Martin and Matthias were always fun and full of humor.

I would also like to thank the members of my thesis committee, Deborah Estrin, Thomas Gross, and Steve McCanne, for their time, availability and helpful feedback, and Emre Telatar for presiding over the defense. I further thank Steve McCanne for his recent advice, help, and availability to discuss my next steps.

Mostly likely due to our having been collectively labelled as the "networking people" (the meaning of which is quite vague, but clearly has suspicious overtones in a signal processing group), Guillermo and I were initially assigned to the same office. This turned out to be a great match, as illustrated by our common appreciation for advanced forms of untidiness, and by Guillermo's patiently humoring my repeated failures at uttering the same handful of Spanish sentences every day for approximately four years. I would also like to thank Andrea and Thibaut for their company in exile at PSE, and everyone at LCAV for contributing to the a collaborative, stimulating and all-around pleasant atmosphere of the lab. Also I should acknowledge the work of Jocelyne without whom the entire laboratory would long-ago have been submerged in chaos.

During the course of my graduate studies I was very fortunate to spend two summers at UCLA working with Deborah Estrin and her group. I would like to thank Deborah for hosting me and finding time in a very busy schedule to be a de-facto advisor during those two summers. I am also grateful to all the members of the LECS lab for their openness and willingness to bounce around ideas.

This dissertation work was influenced by and grounded in a lot of practical work with low-power wireless platforms. I am very grateful to Rémy Blank and Roger Meier for sharing their expertise in low-level embedded software development when I started working on the port of TinyOS to the Tinynode platform, and to Maxime Muller for his important contributions to solidifying the first version of the XE1205 driver. Working with Thomas Schmid on a number of projects since 2004 has also been a great pleasure.

Finally, I would like to thank my family for always supporting me in my various undertakings, up to and including this one, and I am sure in the next ones as well. And most of all, thank you Pansy for your selfless patience, support, and constant supply of encouragement and optimism.

Contents

Abstract i

Resumé ii

Acknowledgements iii

List of Figures ix

List of Tables xii

1 Introduction 1
- 1.1 From Next-hop Node to Next-hop Set . . . 2
- 1.2 Background and Motivation . . . 4
  - 1.2.1 Multi-hop Wireless Networks . . . 5
  - 1.2.2 Anycast Forwarding . . . 6
  - 1.2.3 Routing with Anycast Forwarding: State of the Art . . . 7
- 1.3 Anypath Routing Overview . . . 9
- 1.4 Packet Combining . . . 13
- 1.5 Summary of Contributions . . . 13
- 1.6 Dissertation Overview . . . 15

2 Background and Related Work 17
- 2.1 A Brief Review of Multipath Routing . . . 17
- 2.2 MAC-layer Anycasting . . . 19
- 2.3 Opportunistic Routing . . . 20
  - 2.3.1 Three Forms of Opportunism . . . 20
  - 2.3.2 Selecting and Arbitrating among Multiple Receivers . . . 22
- 2.4 Low-Power Link Layers . . . 23
  - 2.4.1 Synchronous Duty Cycling: Clustered Synchronization . . . 26
  - 2.4.2 Synchronous Duty Cycling: Individual Synchronization . . . 27
  - 2.4.3 Asynchronous Duty Cycling . . . 28
  - 2.4.4 Comparison of Duty Cycling Schemes . . . 30
- 2.5 Packet Combining . . . 32
- 2.6 Summary . . . 33

3 Network Metrics and Models **35**
3.1 Overview and Notations 36
3.2 Spatial and Temporal Traffic Patterns 37
3.3 Link Layer Aspects 38
3.3.1 Reliability and Error Models 38
3.3.2 Media Access and Carrier Sensing 41
3.3.3 Retransmissions 42
3.3.4 Link Layer Models 43
3.4 Link and Path Cost Metrics 44
3.4.1 Always-On Link Costs 45
3.4.2 Duty-Cycled Link Costs 46
3.5 Single-Path Routing and Bellman-Ford 51
3.6 Summary 53

4 Anycast Forwarding **55**
4.1 Link-Layer Anycast 57
4.1.1 Candidate Relay Set 57
4.1.2 Sender-Driven versus Receiver-Driven Forwarding 59
4.1.3 Effective Relay Selection 60
4.2 Anycast Link Cost 63
4.3 Remaining Path Cost 65
4.4 Anycast Forwarding with Always-On Links 72
4.4.1 Anycast Link Costs 72
4.4.2 Remaining Path Cost 74
4.5 Anycast Forwarding with Asynchronous Duty-Cycling 75
4.5.1 Anycast Link Cost 76
4.5.2 Remaining Path Cost 81
4.6 Anycast Forwarding with Synchronous Duty Cycling 82
4.6.1 Anycast Link Cost 82
4.6.2 Remaining Path Cost 84
4.7 Summary 85

5 Anypath Routing **89**
5.1 Physical Cost Models 91
5.1.1 Always-on Networks 92
5.1.2 Duty-Cycled Networks 95
5.2 Shortest Anypath Routes 96
5.2.1 Cost of Anypath Routes 98
5.2.2 Shortest Anypath Routes 100
5.2.3 Structure of Shortest Anypath Routes 105
5.3 Computing the Cost of Acyclic Anypath Routes 109
5.4 Finding Shortest Anypath Routes 113
5.4.1 Bellman-Ford Anypath Routing with Physical Costs 114
5.4.2 Reducing the Search Space 119
5.5 Summary 122

6 Performance and Variations of Anypath Routing 125
6.1 Evaluating Anypath Routing 126
6.1.1 On Policy vs Mechanism 126
6.1.2 Methodology 128
6.1.3 Performance Evaluation 130
6.2 Suboptimal Anypath Routing 132
6.2.1 ERS-any 133
6.2.2 ERS-all 136
6.2.3 Anypath Route Costs with ERS-any and ERS-all 139
6.3 Anypath Routing Performance 141
6.3.1 Diameter Scaling 142
6.3.2 Density Scaling 144
6.4 Suboptimal Anypath Routing Performance 145
6.4.1 Performance of ERS-any 145
6.4.2 Performance of ERS-all 147
6.5 Anypath Routing with Single-Path Metrics 154
6.6 Robustness and Stability 155
6.6.1 Time-varying Network Models 157
6.6.2 Quantifying Robustness and Stability 159
6.6.3 Route Disconnections and Dead Ends 162
6.6.4 Route Dilation 165
6.7 Summary 166
6.8 Appendix: Computation of Remaining Path Costs 167

7 Protocol Design and Prototype Implementation 171
7.1 Overview 171
7.2 Protocol Design 173
7.2.1 Hybrid Anycast Forwarding 173
7.2.2 Network-Layer Anycast 175
7.2.3 Effective Relay Selection and Arbitration 177
7.3 XE1205 Radio and TinyNode Platform 178
7.3.1 Platform Overview 178
7.3.2 XE1205 Radio Device Driver 181
7.4 Architecture and Implementation 183
7.4.1 Scheduled Link Layer 184
7.4.2 Anycast Link Abstraction 186
7.4.3 Network Layer 187
7.5 Testbed Evaluation 188
7.5.1 Testbed Characteristics and Experimental Setup 188
7.5.2 Experimental Setup 192
7.5.3 Results 193
7.6 Summary 197

8 Multi-hop Error Correction with Packet Combining 199
8.1 Overview 201
    8.1.1 Examples 202
    8.1.2 Comparison with FEC 204
8.2 Channel Measurements 205
    8.2.1 Bit-level Error Measurements 206
    8.2.2 Sources of Packet Loss and ETPD 209
8.3 Decoding and Merging 211
    8.3.1 Linear Block Codes 211
    8.3.2 Packet Encoding and Decoding 212
    8.3.3 Packet Merging 213
    8.3.4 When to Send a Plain or Parity Packet 216
    8.3.5 Extensions 217
8.4 Implementation 218
    8.4.1 CPU Overhead 221
8.5 Evaluation 222
    8.5.1 Single-hop with Retransmissions 222
    8.5.2 Multi-hop Routing 225
    8.5.3 Flooding 227
    8.5.4 Breakdown of Combining Attempts 228
    8.5.5 End-to-end Performance 229
8.6 Summary 230
8.7 Appendix: Single-Hop Analysis 231

9 Conclusions 233
9.1 Limitations and Open Problems 233
9.2 Future Directions 234
9.3 Discussion and Outlook 236
9.4 Summary 237

Bibliography 238

Curriculum Vitae 250

List of Figures

1.1 Unicast and Anycast Forwarding. 2
1.2 Anypath route. 3
1.3 Illustration of anycast forwarding. 6
1.4 Mis-match of single-path metrics with anypath routing. 8

2.1 Illustration of duty cycling and energy consumption of different radio states. 24
2.2 Two forms of synchronous duty cycling. 26
2.3 Illustration of asynchronous duty cycling. 29

3.1 Comparison of shortest path routing under end-to-end reliability (E2E) and expected transmission count (ETX). 46
3.2 Transmission cost increases with reduced length preambles. 49
3.3 Breakdown of single-path route cost. 52

4.1 Different choices of candidate relay sets. 58
4.2 Illustration of available relay sets (ARS) and effective relay selection (ERS). 61
4.3 Breakdown of anypath route cost. 66
4.4 The remaining path cost depends on candidate relay set $C(i)$. 70
4.5 The remaining path cost depends on sender. 71
4.6 Anycast forwarding gain for increasing candidate relay set size. 73
4.7 For anycast transmission, reducing preamble length and using ARQ is an efficient strategy with asynchronous duty cycling. 77
4.8 Probability of preamble hit p_{hit} for anycast forwarding with asynchronous duty-cycling. 78
4.9 Swoosh! 80
4.10 Anycast forwarding gain $G_{d_{iC(i)}}$ in transmission energy cost for asynchronous duty cycling. 81

5.1 The packet delivery probability ALC is not a physical cost model. . . 94
5.2 Illustration of anypath routes. 97
5.3 An anypath route may contain cycles. 97
5.4 Cost of a trajectory relative to four different anypaths. 99
5.5 Shortest anypath route may contain cycles with a non-physical cost model. 102

5.6 Example of a shortest anypath route that does not include the shortest single-path route. 105
5.7 Asymmetry of shortest anypath routes. 106
5.8 Illustration of sub-anypath routes. 108
5.9 With a non-physical cost model, sub-anypaths of a shortest *acyclic* anypath are not always shortest anypaths. 110
5.10 Recursive computation of anypath route cost. 112
5.11 Illustration of the Bellman-Ford algorithm computing shortest anypath routes from every node to the destination. 116
5.12 The best CRS is not just “closer nodes”. 118

6.1 Tradeoff between path length and protocol overhead. 128
6.2 Remaining path cost depends on ERS policy. 138
6.3 Performance of anypath routing with increasing network size: fixed density, increasing diameter. 142
6.4 Idealized representation of diameter and density scaling. 143
6.5 Performance of anypath routing with increasing network sizes: fixed diameter, increasing density. 144
6.6 Comparison of average shortest anypath cost D^{AP} for different ERS policies. 146
6.7 Comparison of shortest anypath routes computed with ERS-any and ERS-best. 148
6.8 Comparison of ERS-all with ERS-best and ERS-any for transmission-count metric with link delivery probabilities $p = 0.5$. 149
6.9 Comparison of ERS-all with ERS-best and ERS-any for asynchronous duty cycling with energy-cost metric. 150
6.10 Comparison of shortest anypath routes computed with ERS-all and ERS-best. 153
6.11 Average out-degrees with asynchronous duty cycling, for two different duty cycles. 154
6.12 Example network where anypath routes based on single-path metrics are identical to shortest anypath routes. 155
6.13 Comparison of AP and SP-AP routes. 156
6.14 Example of a route projection. 161
6.15 Empirical CDF of N_d, the number of disconnected routes, for networks with 100 nodes. 165
6.16 Empirical CDF of N_d, the number of disconnected routes, for networks with 1000 nodes. 166
6.17 Empirical CDF of the number of routes with dead ends, for networks with 100 nodes. 167
6.18 Empirical CDF of d, the average route inflation, for networks with 100 and 1000 nodes. 168

7.1 Single-path anycast data collection with $1, 2$, and 3 sinks. 176
7.2 Network-layer anycasting with anypath routing. 177
7.3 TinyNode core module 179
7.4 Link and network layer framing. 185

7.5 Testbed topology. 191
7.6 Testbed degree distribution for different transmit powers. 192
7.7 Average packet delivery costs for single-path and anypath routing. . . 194
7.8 Testbed topology. 196

8.1 Packet combining with multi-hop routing. 203
8.2 Packet combining with network broadcasting. 203
8.3 Bit error rate and preamble detection measurements. 207
8.4 Empirical bit-error probability conditional on an error having occurred k bits earlier. Horizontal lines show the average bit-error rate for each data set. Top and center: packets with $BER < 0.05$. Bottom: packets with $BER > 0.05$. 208
8.5 Packet Decoding: Recovering the original packet from two corrupt packets, when one is plain and the other is parity. 213
8.6 Packet Merging: Recovering the original packet from two corrupt packets of same type. 213
8.7 Block diagram of packet combining functions on transmit/receive paths. 219
8.8 Receiver Flowchart. 220
8.9 Single-hop throughput efficiency η as a function of raw throughput efficiency p_d. 223
8.10 Shortcut link measurements and impact of SPaC on multi-hop routing performance. 224
8.11 Shortcut link on a two-hop segment. 225
8.12 CDF of efficiency η_m for single-path routing with retransmissions (SP), with hop-by-hop packet combining only (SP-SPaC-H), and unrestricted packet combining (SP-SPaC). 100-byte packets. 227
8.13 Flooding delivery rate (left) and scatter plot of end-to-end delivery rate in SensorScope (right). 228
8.14 Markov chain model for packet combining over a single-hop unicast link. State transitions happen at each packet (re-)transmission. 231

List of Tables

2.1 Power consumption parameters for different radio transceivers. 23
2.2 Asynchronous duty cycling parameters. 28

3.1 Link-layer models considered in this dissertation. 43
3.2 Summary of link models, associated unicast link costs, and their interpretation. 53

4.1 Summary of anycast forwarding applied to different link models. . . . 86
4.2 Summary of anycast link costs, their properties, and comparison with unicast link costs. 87

5.1 Validity of conditions $\mathbf{H_1}, \mathbf{H_2},$ and $\mathbf{H_3}$ for different network models. . 121

6.1 Empirical probability of having over $\frac{n}{10}$ nodes disconnected, for network sizes $n = 100$ and $n = 1000$, and for different parameters p in either the node or link addition/deletions models. 164

7.1 Comparison of radio transceiver characteristics. 180
7.2 Current consumption and wake-up times. 181
7.3 Duty-cycled power consumption. 181
7.4 Packet delivery rates. 195

8.1 Interaction spans. 202
8.2 Observed bit-error rate as a function of number of preamble bit errors N_e^p, for two different payload lengths L. 211
8.3 Receiver processing actions: a) for a valid packet m, b) for two corrupt packets m and n. 217
8.4 Memory footprint in bytes, as a function of maximum packet size L. . 221
8.5 Worst-case CPU overhead of SPaC functions. The CPU energy used for each function is compared with the number of bits that the radio would transmit to use equivalent energy. 221
8.6 Breakdown of attempted and successful combine operations between decode and merging. 229

Chapter 1

Introduction

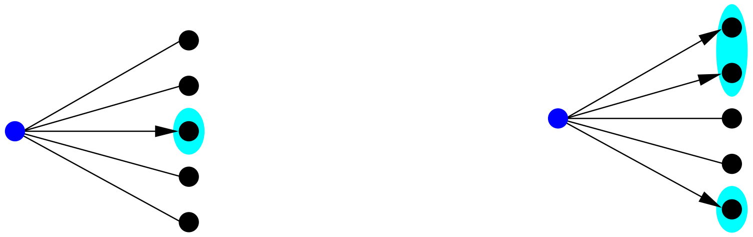
Routing in communication networks has a long history, going all the way back to the shortest-path algorithms proposed in the late 1950s by Ford [47], Bellman[1] [6], and Dijkstra [21]. These algorithms compute the shortest path between a source-destination pair in a graph, and form the conceptual underpinnings of most routing protocols designed and implemented to this date. In single-path routing, each link has a cost, the cost of a route is defined as the sum of its constituent link costs, and the shortest-path route is naturally the path with lowest cost between a source and destination. Within this framework, the fundamental "unit" of communication is the transmission of a packet from the current node to the next hop in the route; this primitive is called *unicast* transmission.

Due to its remarkable simplicity and generality, this combined framework of unicast transmission and single-path routes has worked extremely well, starting from the early days of the ARPANET [53, 75] all the way to present packet networks. Yet, in recent years, a new class of networks has emerged, where it is not a priori evident that single-path routing and unicast transmission form the best framework for multi-hop, point-to-point communication. This is the class of **multi-hop wireless networks**, that consist of nodes connected by wireless links, where some node pairs cannot communicate directly and must use other nodes as intermediate *relays*.

Certainly, it is possible to use single-path routing in a multi-hop wireless network. At the same time, information and communication theory tell us that new architectures will allow significantly improved performance. For example, recent developments in cooperative communication [1] show how the broadcast and multi-point

---

[1] Bellman and Ford independently proposed a similar algorithm. Their names became jointly associated with it, and it is now known as the "Bellman-Ford" algorithm. An enlightening history is found in [18].



Unicast forwarding
Point-to-point links.
Next hop is a single neighbor.
k possible next-hops.

Anycast forwarding
Point-to-anypoint links.
Next hop is **any node(s)** in a **set of neighbors**.
2^k possible next-hop sets.

Figure 1.1: With unicast forwarding, a transmission is sent from one node to a neighbor. With anycast forwarding, a transmission is sent from one node to any node in a set of neighbors. This set is called the **candidate relay set**. The number of possible candidate relay sets is exponential in the number of neighbors k.

nature of a wireless network can be exploited to provide **spatial diversity** [20] in a new way, and how this diversity ultimately can translate into increased throughput or reliability. Unfortunately, the proposed schemes often make assumptions that do not match available technology, such as full-duplex transceivers, hardware that can repeat and amplify an analog signal, or exact and instantaneous knowledge of fading coefficients.

1.1 From Next-hop Node to Next-hop Set

The starting point of this thesis is a simple observation that also derives from a form of spatial diversity: *in a wireless network, it is often less costly to transmit a packet to any node in a set of neighbors than to one specific neighbor.* One instance of this observation is with unreliable links, where the probability that at least one node out of n neighbors receives a transmission increases with n.

The communication primitive of sending to any node in a set is called anycasting, and has been used previously at the network and application layers, for example to provide load-balancing or transparent server replication. In this work, we are concerned with anycasting at the link layer, across a single hop, and we call this primitive **anycast forwarding**. We illustrate anycast forwarding and compare it with unicast forwarding in Figure 1.1.

Motivated by the observation that anycast forwarding is often less costly than unicast forwarding, this dissertation introduces a generalization of single-path routing called **anypath routing**. Anypath routing marks a fundamental departure from

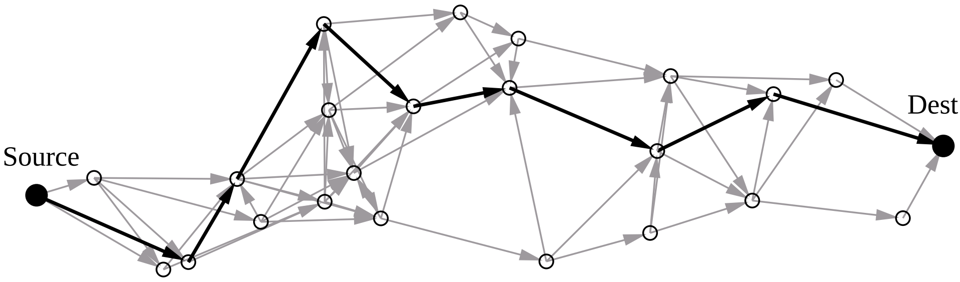


Figure 1.2: An anypath route is a directed graph connecting a source to a destination, and induced by the choice of candidate relays at each node. A packet may traverse the anypath route according to a number of different trajectories, depending on the outcome of the successive anycast forwarding hops. This figure shows an anypath route, with a possible trajectory through the route highlighted in bold.

existing routing frameworks in the following ways:

1. The next hop to reach a destination is a **set** of neighbors rather than a single neighbor. This set is called the candidate relay set (CRS).

2. The underlying communication primitive is **anycast forwarding**, where a node transmits a packet to *any* node in the candidate relay set.

3. The notion of single-path route is generalized to that of **anypath route**, which is the union of all possible trajectories (e.g., sequences of nodes) along which a packet may travel from a source to a destination, given an assignment of candidate relay set to each node. Figure 1.2 shows an example anypath route and a possible trajectory across it.

In single-path routing, computing the shortest single-path route from every node to the destination requires identifying, at each node, which neighbor is the best placed to reach the destination. This dissertation addresses the corresponding problem when using anycast forwarding, namely, the **shortest anypath routing** problem: how to assign a candidate relay set at each node for a given destination, such that the expected cost of forwarding a packet to the destination is minimized. The key is the following tradeoff: on the one hand, by increasing the number of candidate relays, we decrease the cost to send to *any* of these relays. On the other hand however, some of these candidate relays may not be as well positioned as others along the straightest path to the destination. Therefore by increasing the number of our candidate relays, we increase the likelihood of a packet veering away from the shortest-path route, and ultimately we may even introduce loops in our routing topology.

Shortest anypath routing is different from classical shortest single-path routing in at least the following two respects. First, there are 2^k (where k is the number of neighbors) possible candidate relay sets to choose from, in contrast with k possible next-hops for single-path routing. Second, the cost of an anypath route depends on the cost of a large number of possible trajectories that arise from the choice of candidate relay sets, and on the each trajectory's probability of being used.

These differences might appear to indicate that shortest anypath routing will be significantly more complex, and harder to achieve in a practical setting than single-path routing. A core contribution of this dissertation is to show that this is not the case, and specifically, that:

1. Shortest anypath routes can be computed in a distributed setting using a generalized form of the Bellman-Ford algorithm, with an upper bound on convergence time that is equal to that of single-path Bellman-Ford.
2. Under a set of technical conditions that hold true in many cases of practical interest, the search space of candidate relay sets can be reduced from size 2^k to size k.

The shortest anypath routing algorithm is expressed within a general framework, that can accommodate a number of different network and cost models. We show how existing anycast forwarding mechanisms can be mapped into this framework, and further introduce new anycast forwarding mechanisms for low-power networks, that also benefit from shortest anypath routing.

Thesis statement

This dissertation proceeds from the following thesis statement:

> Anycast forwarding is a practical and general way to exploit spatial diversity for communication across multiple wireless hops. Using anycast forwarding effectively requires revisiting the notion of a route, its associated metrics, and the protocols used to compute these route.

1.2 Background and Motivation

This dissertation is motivated by the confluence of two factors. The first is the emergence of multi-hop wireless as an increasingly important network substrate, that is fundamentally different from a wired network. The second is that link-layer anycast forwarding is a relevant and useful primitive in wireless networks, primarily because the wireless medium is broadcast. We outline in this section two important applications of multi-hop wireless networks, and introduce the primitive of anycast forwarding. While we introduce novel anycast forwarding schemes and an encompassing framework in this dissertation, we are not the first seeking to exploit some form of anycast forwarding in a multi-hop wireless context. On the contrary, a number of prior proposals have been made. The last part of this section briefly overviews these attempts at integrating routing with anycast forwarding, and and describe why none of them allow to fully realize the benefits of the anycast forwarding primitive.

1.2.1 Multi-hop Wireless Networks

A multi-hop wireless network consists of nodes connected by wireless links, where some node pairs cannot communicate directly and must use other nodes as intermediate *relays*; such a topology makes these networks fundamentally different from existing cellular networks or wireless LANs that are ubiquitous today. While multi-hop wireless networks are not new (e.g., the DARPA packet radio network [48] that was operated over 20 years ago), they have emerged in recent years as a growing class of networks, driven by at least two broad applications: mesh networks for internet access, and sensor networks for monitoring and controlling physical environments.

Mesh networks

Mesh networks are a first example of multi-hop wireless networks. A wireless mesh network extends Internet connectivity by placing multiple wireless nodes to cover an area. One (or more) of these nodes has a back-haul connection to the Internet and serves as a gateway; the other nodes route packets to and from the gateway(s). Today's mesh networks typically use 802.11 [78] radio interfaces; the newer IEEE 802.16 ("WiMax") standard is likely to see increasing use in coming years. Mesh networks are different than wired networks in at least two aspects that are directly relevant to the design of suitable routing protocols. First, the wireless media is broadcast, meaning that a transmission from one node to a neighbor can also be received by other neighbors, and can interfere with other transmissions. Second, wireless links fluctuate over time due to fading, noise, and changes in the physical environment. These fluctuations are both frequent and unpredictable; being due to external factors they happen independently of network congestion.

Sensor networks

Sensor networks are a second example of multi-hop wireless networks. Devices in a sensor network combine computation, sensing, and wireless communication, and serve to monitor, control and provide real-time visibility into the physical world. Example applications include management of physical resources [2], building automation [109], remote meter reading, and environmental monitoring [100] [39]. Similar to mesh networks, nodes relay packets for others, usually in order to reach a gateway to wired back-end systems.

Sensor networks share the two characteristics of mesh networks described above. Furthermore, many sensor networks face the added requirement of extremely low-power operation. Indeed, most applications employ very low data rates, and are only realistic if nodes can operate untethered and unattended for years at a time. This impacts routing protocols because it requires turning off the radio as frequently as possible, due to the radio being the dominant consumer of energy in such a device (including during idle listening).

1.2.2 Anycast Forwarding

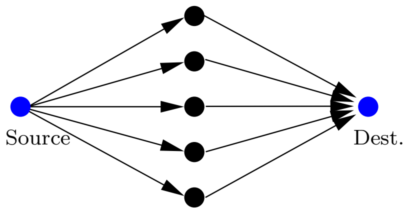


Figure 1.3: Illustration of anycast forwarding in a network with a source, a destination, and 5 intermediate nodes. If links are unreliable and independent, the probability of a packet arriving from the source to the destination is greater if *any* intermediate node can relay packets from the source than if the source selects in advance a next hop.

The use of link-layer anycasting with wireless networks was, to our best knowledge, first put forth by Larsson in 2001 [61], in the form of a joint forwarding and MAC layer protocol where data frames are multicast to a set of candidate nodes. Each receiver sends back an ACK, and the sender then issues a forwarding order to the chosen relay. Other anycast forwarding mechanisms have since been proposed [15, 45, 68], with similar goals of improving performance in the face of link unreliability. A very simple scenario in which anycast forwarding would be useful is shown in Figure 1.3, where a source and destination are separated by an array of 5 intermediate nodes.

If the wireless links in this network are unreliable, and if a packet transmitted by the source is received (or lost) independently by each intermediate node, then the probability that any one of these nodes receives a packet is greater than the probability that one specific node receives it. By allowing any of the intermediate nodes to serve as a relay, the overall delivery probability from the source to the destination is reduced, or equivalently, the required number of transmissions to delivery a packet is decreased.

Of course this is a contrived topology, where it is clear that any of the intermediate nodes are *candidate relays* that can be used to reach the destination. But how should we select the candidate relays to reach a particular destination in an arbitrary network?

1.2.3 Routing with Anycast Forwarding: State of the Art

Our work is not the first to ask this question, and several designs combining multi-hop routing with anycast forwarding have proposed schemes to select the candidate relays used by anycast forwarding. We group these into the three following categories:

- Schemes based on **geographic positions** [68,113]**:** Any neighbor that is closer to the destination than the current node (in Euclidean distance) is a candidate relay.
- Schemes based on **single-path metrics** [9,80]**:** Nodes run a single-path routing protocol to compute their shortest-path distance to the destination. Any node that is closer (in shortest-path distance) to the destination than the current relay can be used as the next-hop relay. This approach is therefore conceptually similar to the geographic schemes above, but with a crucial change in underlying distance metric.
- Schemes based on existing **multi-path routing protocols** [45]**:** A multi-path routing protocol provides multiple path choices to reach a destination. As a result, some nodes have more than one possible next-hop toward a destination, and any of these next-hops can be used by the link-layer anycasting mechanism.

The three categories outlined above each have distinct strengths, and each have the advantage, in comparison with unicast-based single-path routing, of using anycast forwarding. At the same time, none of them offers a completely satisfactory solution to the problem of combining anycast forwarding with multi-hop routing. We discuss the strengths and limitations of existing schemes in the remainder of this section.

Geographic anycast forwarding

Given that Euclidean distance is, after all, the measure of physical proximity to a destination, using geographic positions is an attractive and natural way to select candidate relays. However, a large number of experimental studies [13, 32, 108, 114, 116, 118] have shown that radio propagation is highly irregular at small scales. The

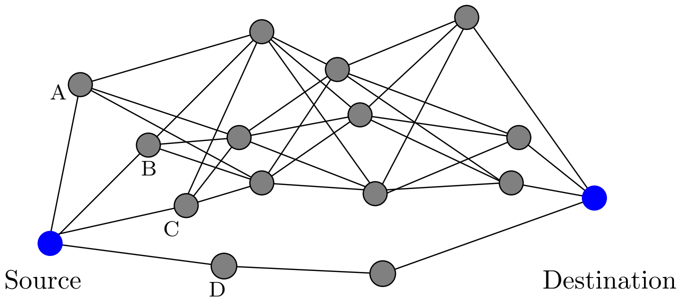


Figure 1.4: Mis-match of single-path metrics with anypath routing. Sending a packet via nodes A, B, or C takes advantage of anycast forwarding, because at each hop in the dense mesh there are three possible next hop candidates. Routing via any of these nodes may be cheaper than through D, even if the route through D traverses fewer hops.

quality and even the existence of a link between two neighbors cannot be determined by the distance between them. Therefore, while advancing geographically is a valid measure of progress at a *global* scale and over large distances, it is by no means a guarantee of good progress at a local scale. In addition, the requirement that nodes know their geographic coordinates is not fulfilled in most existing and planned multi-hop wireless networks[2].

Anycast forwarding with single-path metrics

Single-path routing does not suffer from the problems of position-based routing, since with a suitable link metric, path distances reflect the actual network topology rather than its physical layout. A further advantage is that using single-path routing allows us to leverage well-understood algorithms and protocols. However, the use of single-path metrics to drive the choice of candidate relays does not result in optimal routing choices. The fundamental reason is the following mis-match:

> Single-path routing is based on point-to-point link costs, but anycast forwarding uses point-to-anypoint links.

As a consequence of this mis-match, anypath routing schemes that are based on single-path metrics in general do not compute the shortest anypath routes. An example is shown in Figure 1.4. The source has four neighbors, and must select a subset of these neighbors as the set of candidate relays that may be used to reach the

---

[2] Some authors have proposed to do “geographic” routing using virtual coordinates that reflect actual network connectivity [89] rather than physical location. It may be possible to integrate such approaches with anycast forwarding. We do not explore this avenue in this dissertation.

destination. Let us assume that all links have packet delivery probability $p = 0.75$, and compute delivery probabilities using a single-path metric. The probability of a packet being successfully delivered to the destination when sending via D through the two-node strand at the bottom is $p^3 = 0.42$. The probability of a packet being successfully delivered when going through any 4-node path in the mesh at the top is $p^5 = 0.24$. A single-path metric would therefore lead us to select node D as the sole candidate relay from the source, since A, B, and C each have a lower delivery probability to the destination than the source itself. However, with anycast forwarding, each node in the upper mesh has three possible next-hop relays to its right, and so the probability of delivery across the upper mesh is actually higher than 0.24. Indeed, a simple computation shows that the true delivery probability, when using A, B, C as candidate relays and going through the upper mesh is $(1-(1-p)^3)^4 \cdot p = 0.70$. If our choice of candidate relays is driven by single-path metrics, we would ignore this opportunity, and as a result make a routing decision that provides a significantly lower delivery probability; the single-path metric effectively *disqualifies* relays that in fact should be used.

Anycast forwarding with multi-path routing

The third existing approach is to use a multi-path routing protocol to provide multiple next-hop choices to the anycast forwarding link layer. We overview multipath routing in Chapter 2, but for now, note simply that the original design goal of most multi-path routing protocols (and in particular, of MAODV) is to improve load-balancing, redundancy or failover by providing multiple route choices. Thus there is no a priori indication that running an existing multi-path protocol, whose design is guided by a different objective, is the best way to select candidate relays at each node.

1.3 Anypath Routing Overview

In the following, we outline the key concepts of anycast forwarding and anypath routing, and their application to low-power networks.

Constituent costs

In order to identify the optimal candidate relay set at each node, it is necessary to establish precisely the cost associated with each possible choice. We break this cost down into two constituent components: the **anycast link cost** and the **remaining path cost**. The anycast link cost is the expected cost from the current node to the next hop, while the remaining path cost is the cost from the next hop to the destination. The difficulty in computing these costs is of course that the next hop is not one specific node, but is *any* node in the candidate relay set.

The **anycast link cost** (ALC) is the expected cost of transmitting a packet to any node in a set of neighbors; if that set consists of a single node then naturally it reduces to the regular (unicast) link cost that is used by single-path routing. In a network with unreliable links, the ALC may for example be defined as the probability that a node in the next hop set receives a packet, or the as average number of transmissions until any node in the set receives a packet. Other definitions of ALC are possible and explored further in this dissertation, in particular in conjunction with novel link-layer designs that seek to reduce energy cost and latency of low-power wireless communications.

The **remaining path cost** (RPC) covers the remainder of the cost in an anypath route: it is the expected cost to get from any node in the candidate relay set to the destination. The RPC therefore depends on the cost to reach the destination from each node in the candidate relay set. It also depends on the probability for each node in the set of being actually used as a relay. For example, if each candidate relay is used with same probability, then the RPC is simply the average of the costs from each candidate relay to the destination. In general however, the probability of a packet going through a particular node in the candidate relay set depends on the underlying link delivery probabilities. In consequence, each candidate relay is generally *not* used with same probability, and the RPC is more complex than a simple average. The probability that a particular relay is used depends also on specific policies and mechanisms in the anycast forwarding link layer. These are effective relay selection and relay arbitration. We discuss these aspects next.

Effective relay selection

When transmitting over a broadcast medium, it is possible that a transmission is received by more than one node in the candidate relay set. When this happens, we must choose which of the receivers should be the effective relay, that is, which node(s) then transmits the packet further. We call this the **effective relay selection** policy. One such policy is to choose at random one of the receivers to be the effective relay. Another is to choose the receiver that has the lowest cost to reach the destination. Whenever multiple candidate relays receive a transmission, the relay selection policy thus affects the outcome of anycast forwarding, and consequently the choice of policy has an influence on the remaining path cost. For example, the RPC is higher when choosing an effective relay at random among multiple receivers than when choosing the one with lowest cost to the destination (except of course if all candidate relays have same cost to the destination).

At this point, it may appear surprising to consider policies such as the random selection of effective relay, since selecting the best receiver should always be preferable to selecting one at random. This would be indeed be true if there were no overhead to *execute* the relay selection policy. However, doing so requires reaching agreement on which node(s) actually received the packet, making the selection decision, and communicating this decision, all in a distributed setting where communication is both unreliable and costly. This overhead is not the same for all policies, and so an important feature of our framework is that it encompasses a multitude of relay selection policies, allowing a protocol designer to evaluate the benefits of a given policy in terms of both its protocol overhead, and of the relay selection choices it makes.

An extreme case of relay selection policy is for *all* candidate relays receiving a packet to forward it. In this case, a packet may be duplicated into additional copies that "fork" into separate branches. This policy has the advantage of having little or no overhead to carry out, since any receiver can forward the packet without knowing if other candidate relays received it. However, it is clear that it can lead to a lot of duplication, and thus should be used with precaution. There are still situations where this policy may be interesting, namely when it is very rare that multiple candidate relays receive a packet. One such situation is with the different low-power anycast forwarding schemes developed in this dissertation.

Shortest anypath routing

Solving the shortest anypath routing problem requires us first to define the cost of an anypath route. Because an anypath route is composed of many possible trajectories (including, possibly, trajectories with forks and duplicated packets), the expected cost of an anypath route must reflect the cost of each individual trajectory as well as the probability of it being used to traverse the route. The cost of each possible trajectory depends not only on the sequence of nodes that it traverses, but also on the candidate relay sets of each node in the trajectory. The probability of a given trajectory being taken depends not only on individual link properties (such as packet delivery probability), but also on specifics of the anycast forwarding technique being used. For example, the effective relay selection policy plays a role in determining which receiver forwards a packet (when there are multiple candidate relay have received it), and so the choice of policy influences the probability that a given trajectory is taken.

The idea that the cost of an anypath route is determined by the cost and probability of each possible trajectory across it is quite intuitive. However, how to effectively *compute* is not so immediate. One possible approach is to enumerate each feasible trajectory explicitly and to compute its probability and cost. But such an enumeration would be tedious and complex, and would not lead to a distributed implementation. We define instead an algorithm that proceeds recursively backward from the destination to the source, without needing to explicitly enumerate trajectories, and show in Proposition 5.9 that it computes the cost of an anypath route in a number of iterations bounded by the length of the longest trajectory.

While finding a way to compute the cost of a given anypath route is a necessary first step, it does not yet solve the harder problem of computing shortest anypath routes. The key to do so is to show that *sub*-routes of a shortest anypath route are themselves shortest anypath routes. This property (which holds trivially for single-path routes) allows us to compute and reason about shortest anypath routes in a recursive way. A distributed formulation of the algorithm computing shortest anypaths then follows quite directly. This formulation is a generalization of the distributed Bellman-Ford algorithm; the crucial difference with single-path Bellman-Ford is that the involved cost metrics are the anycast link cost and remaining path cost. In the general case, the algorithm must search among 2^k possible subsets of nodes (for a neighborhood of size k). Fortunately, it is possible to reduce this search space to size k in many cases of practical interest.

While anypath generalizes single-path routing, shortest anypath routes have certain properties that are different and somewhat surprising. For example, shortest anypath routes are usually not symmetric (even when the underlying links are symmetric), and the shortest anypath and shortest single-path routes between two nodes may be disjoint.

Application to low-power wireless networks

In addition to establishing an algorithmic basis and design framework for anypath routing, our work also proposes novel anycast forwarding link layers that exploit spatial diversity differently than previously. In particular, we explore the space of **low-power wireless networks**, and show how anycast forwarding can be used to reduce both energy cost and latency when nodes spend most of their time sleeping. One example is when low-power nodes sleep and wakeup randomly and without synchronization. In this setting, the probability that *any* node in a set of neighbors is awake during a packet transmission is greater than the probability that a given neighbor is awake. This can be exploited to reduce the energy cost of packet transmissions by a factor of 3 or more, depending on the number of candidate relays.

Our practical validation of anypath routing takes place within this domain of low-power communication. We design an anypath routing protocol and low-power anycast forwarding link layer, and implement this design on a low-power wireless platform.

The fact that this platform is extremely resource constrained (both in memory and computation) confirms that anypath routing is practical. An experimental evaluation using a wireless testbed shows that anypath routing delivers performance gains that are on par with those predicted by theory and simulation.

1.4 Packet Combining

While anycast forwarding is a powerful and general mechanism to exploit spatial diversity in a multi-hop wireless network, other techniques are possible. This dissertation investigates one such other technique with the design, implementation, and experimental evaluation of an error-correction system based on *packet combining*. Nodes buffer corrupt packets (rather than dropping them upon reception), and when two or more corrupt versions of a packet have been received, a packet combining procedure attempts to recover the original packet from the corrupt copies.

Classical forward error correction (FEC) operates on point-to-point basis, correcting (when possible) communication errors between any two given nodes at a time. Packet combining does not only this, but additionally operates over multi-point interactions such as multi-hop routing or broadcasting. For example, a corrupt packet overheard from a transmission two hops upstream can be used to later recover errors as the transmission travels down the route. The net result is the opportunity to recover from more errors than in classical error correction by exploiting a larger number of input packets. The scheme also has the very practical advantage that it does not transmit redundant error-correction overhead on good links, and it does so without requiring estimates of channel conditions.

Packet combining exploits the broadcast medium and spatial diversity of a multihop wireless network by using corrupt packets overheard at *any* node, in addition to the next-hop destination of the packet itself. It is therefore complementary to anypath routing and fits naturally into the framework of anycast forwarding.

1.5 Summary of Contributions

The main contributions of this dissertation are:

- **A framework for anycast forwarding.** We define a framework for the design and modelling of anycast forwarding primitives. Modelling an anycast link layer within this framework is achieved by establishing its anycast link cost and remaining path cost. The framework is general, encompasses a wide variety of link layers, and can account for all key aspects of anycast forwarding such as effective relay selection, receiver or sender-driven forwarding, and duplicate forwarding.

- **Novel anycast forwarding mechanisms for low-power wireless link layers.** We show how anycast forwarding can increase energy efficiency or decrease latency in low-power wireless networks, and introduce new anycast link-layer mechanisms to reduce preamble length or latency with low-power, duty-cycled wireless links.

- **Shortest anypath routing.** We define anypath routes, their cost, and formulate an algorithm that finds the shortest anypath route between a source and a destination. This algorithm generalizes the single-path Bellman-Ford algorithm, and can be operated in a distributed setting. Its correctness is proven in Theorem 5.10, and its convergence time has the same upper bound as the single-path Bellman-Ford. We also show how the search space of candidate relay sets can be reduced from size 2^k to size k (where k is the number of neighbors) in many cases of practical interest.

- **Relay selection and arbitration.** When multiple candidate relays receive a packet, coordinating and arbitrating to designate the effective relay have costly protocol overhead. We show how it is possible to reduce this overhead at the cost of using suboptimal routes, investigate this tradeoff, and show cases where exploiting this tradeoff is of practical relevance.

- **Implementation and validation.** Anypath routing is a general technique applicable to a wide variety of networks. We have validated it in one particular instance, namely *low-power wireless networks*, by designing, implementing, and evaluating an anycast forwarding link layer and anypath routing protocol. Evaluation on a 50-node testbed demonstrates that anypath routing is robust, stable, and increases energy efficiency of low-power nodes by a factor of up to 300% over the equivalent system using single-path routing.

- **Packet combining.** Using packet combining for error correction is another expression of spatial diversity, that integrates very naturally with anypath routing. Various forms of packet combining were already proposed and analyzed by others; our contribution here is the first design, implementation, and evaluation of a multi-hop packet combining error correction scheme in an experimental setting.

1.6 Dissertation Overview

This dissertation proceeds as follows. In Chapter 2, we review background material and related work. This includes work related to anycast forwarding and anypath routing, as well as existing low-power wireless link-layers that our new anycast forwarding mechanisms are based upon. Chapter 3 describes some simple network models that we use throughout this thesis, and their associated cost metrics.

At this point the stage is set. Chapter 4 introduces a general framework for anycast forwarding, and introduces new anycast mechanisms specifically geared towards low-power networks. Chapter 5 is where everything comes together: we define anypath routes, their cost, and formulate a shortest anypath routing algorithm based on Bellman-Ford. Then in Chapter 6 we explore variations on anypath routing and conduct a simulation-based study of the tradeoffs involved. Chapter 7 presents our prototype implementation, and Chapter 8 presents our packet combining error-correction scheme. We finally conclude in Chapter 9.

Chapter 2

Background and Related Work

This chapter gives an overview of related work and contrasts it with the contributions of this dissertation. We first survey multipath routing, that superficially resembles anypath routing, but has entirely different motivations and mechanisms.

Anypath routing spans both link-layer forwarding mechanisms and network layer routing decisions. Accordingly, related work includes both papers that specifically address either link-layer anycasting or routing, and those that combine both aspects.

We also survey existing link-layer methods for ultra-low power operation of low-rate wireless networks. These are not related to anypath routing per se, but form the basis of the low-power anycast forwarding models of Chapter 4, and of the system designed in Chapter 7. Finally, the last section of this chapter discusses work related to our packet combining error-correction method.

2.1 A Brief Review of Multipath Routing

The idea of using multiple paths to route packets to a destination has been around for decades, going all the way back to the early work of Maxemchuk [72]. In a very general sense, multipath routing techniques seek to generate and exploit *path diversity* in a network.

Anypath routing can thus be viewed as a form of multipath, since it allows successive packets to traverse multiple paths. The fundamental difference between anypath and multipath routing is that anypath seeks to generate spatial diversity *at local scales* in order to exploit a link-layer mechanism (anycast forwarding); multipath usually seeks to generate diversity at the scale of entire paths. This is reflected in the way most multipath routing protocols operate: by discovering single-path routes (perhaps with some constraints, such as routes being disjoint), and assembling them to obtain a multi-path route. In contrast, an anypath route is a single structure that is computed as such, and not a juxtaposition of single-path routes.

Surveying the vast body of work in multipath routing is beyond the scope of this dissertation; we rather seek to show a few representative examples to highlight this fundamental difference between anypath routing and multipath.

Multipath routing in wired networks

Dispersity routing, proposed by Maxemchuk in his dissertation work [72], is a form of multipath routing that spreads the data from a source over several paths through the network by transmitting on lower rate periodic channels on each of the paths. It was initially investigated as an alternative to adaptive routing in packet networks networks since both mechanisms equalize the load on a network [71]. The two types of dispersity routing are *non-redundant* and *redundant* dispersity routing. In the non-redundant form, a message is divided into submessages that are sent across different paths. All submessages are required at the destination in order to reconstruct the original message. In the redundant form, submessages contain additional redundancy obtained via the use of erasure codes. The destination can then reconstruct the original message using fewer submessages, giving higher resilience to packet loss, but the total number of transmitted bytes is higher. More recent work [73] applied dispersity routing to ATM networks, where the spreading of data over multiple paths can be used to decrease either the fraction of capacity used on any given path, or the latency to transmit a packet.

Dispersity routing does not address the problem of *finding* the multiple paths, but rather focuses on characterizing the performance gains that arise under idealized multipath route abstractions. These abstractions assume *disjoint* paths, and the analysis can simply consider N independent channels from the source to the destination, without modeling the topology itself. Note that path disjointness is not a relevant criterion to anypath routing; on the contrary if paths are disjoint, each node (except the source) has a single next hop, and the gains from anycast forwarding cannot be realized.

Another example of multipath routing is the work of Zaumen and Garcia-Luna-Aceves [112]. This work defines a routing algorithm that computes the multipaths containing all paths from the source to the destination that are guaranteed to be loop-free at every instant. Our definition of anypath route in Chapter 5 is similar to theirs, but our notion of *shortest* anypath routes is different, because our cost model is designed to reflect the use of anycast forwarding.

Multipath routing can also be used in some Internet routing protocols. For example, the Open Shortest Path First (OSPF) supports a mode of operation known as equal-cost multipath (ECMP) routing, where a router can maintain more than one next-hop for a given destination, using different paths of equal length. Not surprisingly, the considerations for choosing the effective next hop among the multiple candidates are very different to those that we have in a wireless network (see [99]), centering around the difficulties of using network debugging utilities (e.g., `ping` or `traceroute`) or the problems arising when different paths have differing maximum transfer unit (MTU) sizes.

Multipath routing in wireless networks

Closer to this dissertation, different multipath schemes have been proposed for routing in wireless networks. Two representative examples are On-Demand Multipath Distance Vector Routing in Ad Hoc Networks (AOMDV) [70], and braided multipath routing [31]. AOMDV is used by the opportunistic routing protocol of Das outlined in Section 2.3 below. Braided multipath routing is targeted at low-power wireless networks, and is proposed with the goal of increasing resilience to node failures. Both of these protocols seek to find paths with some degree of node-disjointedness or link-disjointedness in the paths that they find. Both are also inherently based on single-path routing concepts, that are assembled to proved multiple different routes. Again, this highlights again the orthogonality of multipath routing with anypath.

Another example of multipath routing in a wireless network is the work of Srinivas and Modiano [96], who propose an algorithm that finds minimum-energy k link- or node-disjoint between two nodes. For link-disjoint paths, a node can have multiple outgoing edges, and their scheme takes into account the energy savings that are realized in this case thanks to the broadcast nature of the wireless medium.

2.2 MAC-layer Anycasting

The general idea of MAC-layer anycasting has already been proposed and motivated by others, albeit always in one specific form related to improving reliability or throughput with unreliable links. Larsson [61] proposed a joint forwarding and MAC layer protocol where a data frame is multicast to a set of candidate nodes. Each receiver sends back an ACK, and the sender then issues a *forwarding order* to the chosen relay. Choudhury and Vaidya [15] propose a similar mechanism, but with the main difference that the anycast decision is made *before* transmission; in this case there is no need for the ACK and forwarding order of Larsson's scheme.

These works focus on the link-layer *mechanisms* necessary to implement anycast forwarding. They assume that the network layer maintains a list of possible next-hop candidates (e.g., with a multi-path routing protocol), that is provided to the link layer, and do not propose specific strategies for the selection of these candidates by the routing protocol

2.3 Opportunistic Routing

An opportunistic routing protocol exploits anycast forwarding by providing each node with multiple candidate next-hops to reach a given destination. We shall see in the remainder of this dissertation that our proposed scheme of anypath routing is *one form* of opportunistic routing; others exist and have been previously proposed. We outline these here, distinguishing as in the introductory chapter between schemes using geographic positions, schemes based on single-path routing, and schemes based on an existing multi-path routing protocol.

When multiple nodes receive a packet, it is also necessary to decide which one should relay it, and to communicate this decision. We discuss existing approaches to this problem at the end of this section.

2.3.1 Three Forms of Opportunism

Opportunistic Routing using Geographic Positions

Using geographic positions is a natural way to exploit anycast forwarding: a node transmits a packet, and the receiver which makes most geographical progress toward the destination is selected as the next relay. Multiple schemes have been proposed and evaluated in theory and simulation; we briefly outline these here. Unfortunately, these schemes have not been validated in an experimental system, nor have the the design tradeoffs required to take these schemes into a real system been fully evaluated.

Zorzi and Rao's work on Geographic Random Forwarding (GERAF) [68] is one of the first opportunistic routing proposals, and uses geographic forwarding. The authors analyze the average progress that a packet makes when the effective relay is selected based upon geographic distance to the destination, under the assumption of nodes being uniformly distributed in surface. The connectivity model is highly idealized, and so the difficulties of geographic forwarding that we outlined in the introduction do not surface in this work. The authors also propose a recursive method to compute the expected path length of a packet routed using their scheme. It is interesting to note this recursion can be expressed in a *forward* manner, from the source to the destination. In contrast, computing the expected path cost with anypath routing is more complicated, and we must recurse *backwards* from the destination to the source, (see Chapter 5, p. 109), because the actual progress of the packet depends on the costs of the nodes ahead of it. With geographic routing, these costs are known a priori since they are simply the Euclidean distance to the destination.

Other related work in this area includes the modeling and analysis of Shah et. al. [92] [91], as well as HARBINGER, a scheme that uses both packet combining and geographic opportunistic routing.

Opportunistic Routing using Single-Path Routing

Opportunistic routing can also be used in conjunction with a single-path routing metric. In this case, the selection of relay by anycast forwarding is driven by the shortest-path distance from each node to the destination.

ExOR [9], proposed by Biswas and Morris, is one such scheme. It is notable for being the first working instance of opportunistic routing, and demonstrates the applicability of opportunistic routing to real systems. ExOR increases throughput of 802.11b multi-hop wireless networks by choosing the effective next-hop relay *after* the packet has been transmitted. The choice of effective relay is driven by single-path routing metrics. In other words, the effective relay is chosen to be the receiver with shortest single-path distance to the destination. If no receiver has lower single-path distance, the sender retransmits.

While such a strategy has the advantage of leveraging well-understood single-path concepts, it does not result in the optimal routing choices. The underlying reason is that shortest single-path route costs do not always reflect route costs when using this anycast forwarding, as we have outlined in the introduction.

This dissertation is not the first work applying anycast forwarding and opportunistic routing to low-rate, lower-power wireless operation. In 2005, Parker and Langendoen proposed an opportunistic routing protocol called Guesswork [80], and conducted a simulation-based evaluation with a focus on low-rate wireless sensor networks. In the simulations, Guesswork is run in conjunction with different low-power MAC layers (specifically, synchronous duty-cycling MAC layers, that we survey in Section 2.4 of this chapter). The authors observe an increase in reliability but not in energy efficiency. This is not entirely surprising, given that Guesswork uses anycast forwarding in essentially the same way as in ExOR: as a means to reduce the expected number of transmissions by having more than one choice of next-hop relay.

This dissertation, in contrast, shows how anycast forwarding can be used in different ways that *directly* enhance energy efficiency or latency with low-rate, low-power links, beyond simply applying existing anycast forwarding techniques that primarily serve to improve throughput in high-rate networks.

We are not the first to argue that single-path metrics are unsuited to opportunistic routing. In particular, Zhong et. al. [115] argue for the use of new metrics, and propose one such metric called Expected Any-path transmissions (EAX). Their motivation is to reduce the size of the candidate relay sets, in order to reduce the protocol overhead of inter-receiver coordination. Using their metric, Zhong et. al. propose to first select candidate relays in the same manner as ExOR (using a single-path metric), and then to prune candidate relay using a heuristic that incorporates the EAX metric. This work is an important step, however it only goes half the way: they show how to compute the expected cost of using an anypath route, but *not* how to find the anypath route with lowest cost. In contrast, our anypath routing algorithm provably computes the shortest anypath route. Furthermore our framework is more powerful, since it accommodates other anycast forwarding metrics than ETX, and can also reflect the use of other policies for effective relay selection.

Opportunistic Routing using Multi-Path Routing

Jain and Das [45] integrate an anycast extension of the 802.11 MAC layer with the multi-path AODV (AOMDV) [70] routing protocol. They observe the same tradeoff as [15] between number of candidates and path length, and modify AOMDV to allow the use of paths up to *one* hop longer than the shortest path. This choice is based on an empirical evaluation, the details of which are not given. Note that the original design goal of most multipath routing protocols (and in particular, of MAODV) is usually to improve load-balancing, redundancy or failover by providing multiple route choices. Thus it would be somewhat surprising that they provide the optimal routes for use with anycast forwarding.

2.3.2 Selecting and Arbitrating among Multiple Receivers

When multiple candidate relays receive a packet, it is necessary to select the one that should relay the packet further. All existing schemes make this selection based on the distance (e.g., geographic or single-path) from each receiver to the destination, and use the one that is closest to the destination. No prior work explores the use of other policies, such as picking one receiver at random to be the relay, or allowing all receivers to relay the packet; we motivate these policies in Chapter 6 and evaluate the tradeoffs involved.

It is important to distinguish between the *policy* for selecting the effective relay, and the *protocol mechanism* to coordinate among nodes so that the intended node does indeed forward the packet, and others do not.

Radio	**XE1205**	**CC2420**	**WCF54G**
Manufacturer	Semtech	TI	LinkSys
Standard	Proprietary (FSK)	IEEE 802.15.4	IEEE 802.11g
Frequency	869 Mhz	2.4 Ghz	2.4 Ghz
Max. Data Rate	76.8 Kbps	250 Kbps	54Mbps
TX Output Power	1 mW	1 mW	31 mW
Radio RX	**46.2** mW	**66** mW	**891** mW
Radio TX	**76.6** mW	**58.4** mW	**1155** mW

Table 2.1: Power consumption parameters for different radio transceivers. Listening and receiving consume comparable energy to transmitting.

It is not easy to find a coordination protocol that does this with acceptable overhead, and consequently a large effort is devoted in all the works cited above to the design and evaluation of relay arbitration schemes. We discuss the underlying ideas behind arbitration protocols in Chapter 6.2.1 (p. 135); the protocol that we develop in the experimental portion of this dissertation is able to *avoid* most of the relay arbitration overhead by in many cases doing away with it altogether and allowing all receivers to forward a packet.

2.4 Low-Power Link Layers

In networks consisting of untethered nodes, in particular wireless sensor networks, nodes must be capable of functioning for extended periods of time without human intervention. Energy (or lack thereof) is usually the main obstacle to achieving long-term, autonomous deployment and energy efficiency is therefore an important design objective in many wireless link layers.

So far we have implicitly considered link layers and networks where radio transceivers are permanently turned on (either listening or transmitting). We call these *always-on* networks. Since the radio transceiver is the dominant energy sink in any wireless device, this always-on strategy is not compatible with long-term energy autonomy, or at least it requires large and costly energy sources (such as heavy-duty batteries or large solar panels).

While we intuitively might expect power consumption to be significantly lower when a radio is in receive mode (either receiving an actual packet, or in idle listening) than when transmitting, receive path circuitry requires signal amplification and significant analog and digital processing, and as a result receive power consumption is comparable to transmit consumption. Table 2.1 illustrates this with three wireless radios. The XE1205 by Semtech and the CC2420 by Texas Instruments are both low-power, low-rate single-chip radio transceivers that are representative of radios

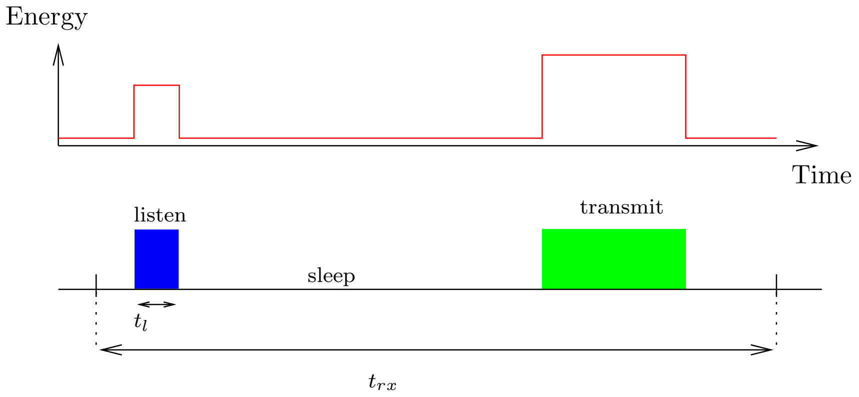


Figure 2.1: Illustration of duty cycling and energy consumption of different radio states. The duty cycle is the ratio t_l/t_{rx}. A node listens to the channel for duration t_l and goes back to sleep because there is no packet to receive; at a later time it transmits a packet. Transmitting is slightly more expensive than listening or receiving; sleeping is significantly less expensive than either transmitting or receiving.

used in wireless sensor networks. The XE1205 uses a proprietary frequency-shift keying physical layer, and the CC2420 implements the IEEE 802.15.4 standard (designed to work with the ZigBee network- and application-layer standard), which uses a direct-sequence spread spectrum physical layer. The LinkSys WCF54G is an IEEE 802.11b/g CompactFlash card with an Atheros chipset.

Since receiving and idle listening have energy cost comparable to transmitting, efficient power management requires more than minimizing the amount of data transmitted: in addition to transmitting as little as possible, nodes should reduce their radio duty cycle and entirely power down the radio as often and as long as possible. We refer to this practice as *duty cycling*, and we define the *duty cycle* as the fraction of time that a radio spends in *listen* mode. We illustrate the concept of duty cycling in Figure 2.1, where a node periodically listens for a duration t_l, every t_{rx} seconds, giving a duty-cycle t_l/t_{rx}.

Note that the duty cycle alone does not completely determine radio energy expenditure, since it does not consider the time spent transmitting. Nonetheless, this definition of duty cycling reflects the fact that in a low rate network, non duty-cycled nodes spend a negligible amount of time transmitting data in comparison with idle listening. Furthermore, this definition is convenient since accounting for transmission time would require precise assumptions on the amount of data transmitted by each node, the traffic and communication pattern, and the network topology.

The goal of operating nodes at very low duty cycles (e.g., 1% or lower) has led to many proposed protocols for media access control (MAC), which we broadly group into two classes: synchronous protocols and asynchronous protocols.

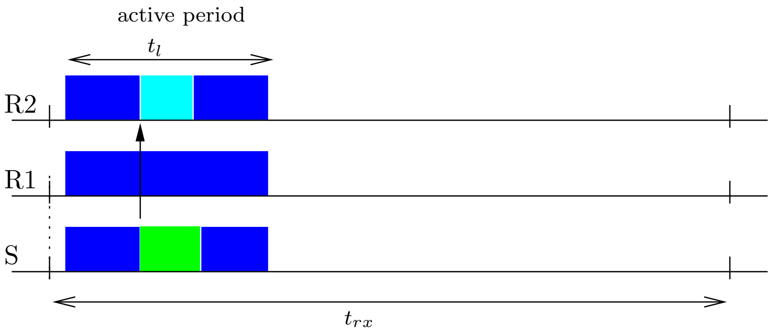
In synchronous protocols, nodes are synchronized such that they know when their neighbors will be awake and listening on the radio. Therefore, a node having a message to transmit to its neighbor can schedule the transmission to occur exactly when the neighbor is listening. Synchronous protocols include those based on time division multiple access (TDMA) protocols [24, 50, 95], beacon-based protocols such as IEEE 802.15.4 [79], S-MAC [110], T-MAC [102] and TRAMA [88].

The second class, asynchronous protocols, takes a different approach. Instead of coordinating duty cycle schedules, nodes periodically wake up, sample the channel, and only start receiving data if they detect channel activity. Depending on the underlying physical layer, this detection can either be based on channel energy or successful symbol decoding. Periodic channel sampling allows a node to conserve energy by keeping its radio off most of the time. Examples include Aloha with preamble sampling [26], B-MAC [84], and WiseMAC [25].

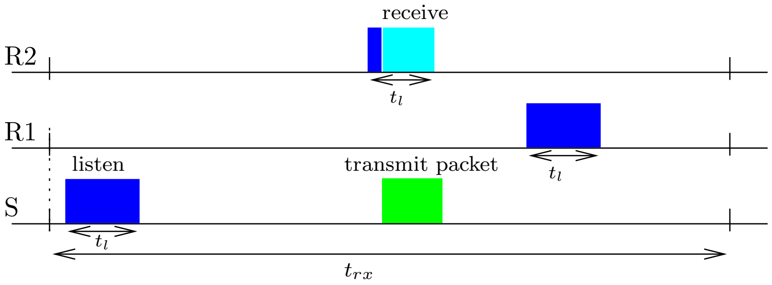
Each class of link layer protocols has its advantages and drawbacks. Synchronous protocols allow nodes to only power the radio on when needed, significantly reducing idle listening. At the same time, these protocols are by nature more fragile and brittle than asynchronous protocols: after they establish a schedule, a node can usually only communicate with other nodes on the same schedule. If for some reason a node fails to synchronism with others or loses its schedule, perhaps due to a link outage or to a software failure, it can become disconnected from neighbors until it has re-established synchronization. Also, establishing and maintaining a synchronized schedule itself has overhead and costs both power and bandwidth.

In contrast to synchronous protocols, asynchronous protocols do not require nodes to learn and maintain schedule state about other nodes: each node wakes up according to a self-determined schedule, and each node can transmit a packet to a neighbor at any time. The only requirement is for all nodes in the network to be configured with a similar duty cycle. This reduced statefulness translates into increased robustness in comparison with a synchronous protocol where faulty or missing schedule information can make a node unreachable. Robustness and simplicity come at a cost, however. Unlike synchronous protocols that send regular data packets, asynchronous protocols must send long, expensive messages to wake up a neighbor.

While they are conceptually similar, the distinction of synchronous versus asynchronous duty cycling is separate from the distinction of scheduled versus random access that we make in Section 3.3.2. Scheduled or random access are medium access control functions serving to manage the use of a shared channel by multiple contending nodes, whereas synchronous versus asynchronous duty cycling serve to manage



(a) Clustered Synchronization



(b) Individual Synchronization

Figure 2.2: Two forms of synchronous duty cycling. Node S has a packet to send to $R2$. (a) With clustered synchronization, all nodes wake up at the same time. Node S can transmit to $R2$ during this active period. (b) With individual synchronization, each node wakes up at a different time. These wakeup times are known to other nodes, and so S knows when to transmit its packet to $R2$.

the communication rendez-vous of nodes which spend most of their time with their radios turned off.

2.4.1 Synchronous Duty Cycling: Clustered Synchronization

The first form of synchronous duty cycling considered here uses *clustered synchronization*, and we shall also refer to it as "clustered duty cycling". As shown in Figure 2.2(a), nodes wake up simultaneously for an interval of duration t_l, and any node wishing to transmit a message then transmits in this interval. The wakeup interval is called the *active* period, and occurs regularly at intervals t_{rx}, so that the overall duty cycle is t_l/t_{rx}. Medium access contention within the active period can be resolved by either CSMA or TDMA mechanisms that we have seen previously.

Clustered duty cycling was originally introduced by the Sensor-MAC (S-MAC) protocol [110] in 2002, and has been since refined leading to its successor SCP [105].

The main difficulty in a clustered synchronization scheme is in getting all nodes on a common schedule. In S-MAC, nodes broadcast SYNC packets at the beginning of the active period, so that other nodes receiving these packets can adjust their clocks to compensate for drift. New nodes that join the network (and nodes that previously had lost synchronization) stay awake until they overhear a SYNC packet allowing them to join the clustered schedule. In theory, the whole network should run the same schedule, but in practice, due to bootstrapping, clock drifts, and the difficulty of reaching consensus across an entire network, multiple synchronization clusters often emerge. In this case, nodes that are at the border between different clusters must wake up for the active period of each cluster they are part of; these nodes therefore have a higher duty cycle than t_l/t_{rx}. A scheme that tries to prevent the emergence of many clusters is presented in [66]; results therein show that while it is possible to somewhat reduce the number of schedules, multiple schedules are difficult to avoid altogether in a large network.

In the S-MAC implementation, the active period t_l is fixed at a fairly large 300 ms, and the duty cycle is controlled by varying t_{rx}. This value for t_l is therefore quite long in comparison with typical settings for asynchronous duty cycling (as we shall see in Table 2.2); it is necessary since all nodes in the network may potentially use this same period for their communications. Note that since decreasing the duty cycle requires increasing t_l, S-MAC trades off latency for energy efficiency.

Optimizations to S-MAC are possible. One example is Timeout-MAC (T-MAC) [102], whose key difference with S-MAC is the introduction of an adaptive duty cycle. With T-MAC, t_{rx} is fixed (at 615 ms). Nodes go back to sleep if there is no traffic activity past some timeout, with the effect of extending or retracting the adaptive period depending on the traffic load.

2.4.2 Synchronous Duty Cycling: Individual Synchronization

Individual synchronization (Figure 2.2(b)) is a different form of synchronous duty cycling where each node adopts its *own* schedule independently and advertises it to its neighbors. Each node keeps track of its neighbors' schedules and can transmit messages at the appropriate time by knowing when the destination will be in an active listening period.

The key difficulty with individual synchronization lies in getting nodes to learn the schedules of each neighbor. One way is for each node to broadcast its schedule at the beginning of each active period, so that a new node can simply listen for a period t_{rx} and learn the schedules of every neighbor. This is probably overkill, and a more efficient approach would be to broadcast schedules only every n periods (at the cost

Platform	Wakeup Period t_{rx}	Listen Time t_l	Duty Cycle t_{rx}/t_l	Packet Time t_{pkt}
Mica2	20 ms	8 ms	28%	15 ms
	85 ms	8 ms	8.6%	15 ms
	135 ms	8 ms	5.5%	15 ms
	185 ms	8 ms	4%	15 ms
	485 ms	8 ms	1.6%	15 ms
	1085 ms	8 ms	0.7%	15 ms
TinyNode	21 ms	2.12 ms	10%	3.7 ms
	79 ms	2.12 ms	2.7%	3.7 ms
	299 ms	2.12 ms	0.8%	3.7 ms

Table 2.2: Example of duty cycle parameters from the link layers of two low-power wireless platforms using asynchronous duty cycling: Mica2 (running at 19.2 kbps) and TinyNode (76 kbps). These parameters come from the TinyOS link layer implementations for these platforms.

of requiring a longer listening time of up to nt_{rx} for new nodes).

2.4.3 Asynchronous Duty Cycling

We now turn to the case of asynchronous duty cycling (also known as Low Power Listening or LPL [84]). Each node awakens periodically for short periods and listens on the radio. If no transmission is detected during this interval t_l, the node goes back to sleep until the beginning of the next sampling interval. We note the wakeup period as t_{rx} and the length of the interval during which the node listens on the channel as t_l ($t_l < t_{rx}$). The duty cycle is then t_l/t_{rx}. Some examples of practical values for t_{rx} and t_l are shown in Table 2.2 for two low-power wireless platforms that have asynchronous duty-cycled link layer implementations in TinyOS.

Although nodes regularly poll the channel with a predefined polling period, their polling times are not explicitly synchronized. To reliably rendez-vous with the receiver, a transmission must be of length t_{rx}, or else it may not coincide with the wakeup time of the receiver. As shown in Table 2.2, the packet duration t_{pkt} is significantly shorter than the wakeup period, and so a packet transmitted without knowledge of the receiver's wakeup schedule is not likely to be received. In typical LPL implementations [23, 84], the packet is preceded by a long preamble (a repeated sequence of bits that is easy to detect, such as 101010101...) of length such that the entire transmission has duration t_{rx}. This preamble is effectively a *wake-up signal*: if a node hears an ongoing preamble during its brief listen time, it stays awake to receive the forthcoming packet. We illustrate asynchronous duty cycling in the diagram of Figure 2.3. Note that wakeup times may be randomly chosen within the t_{rx} intervals,

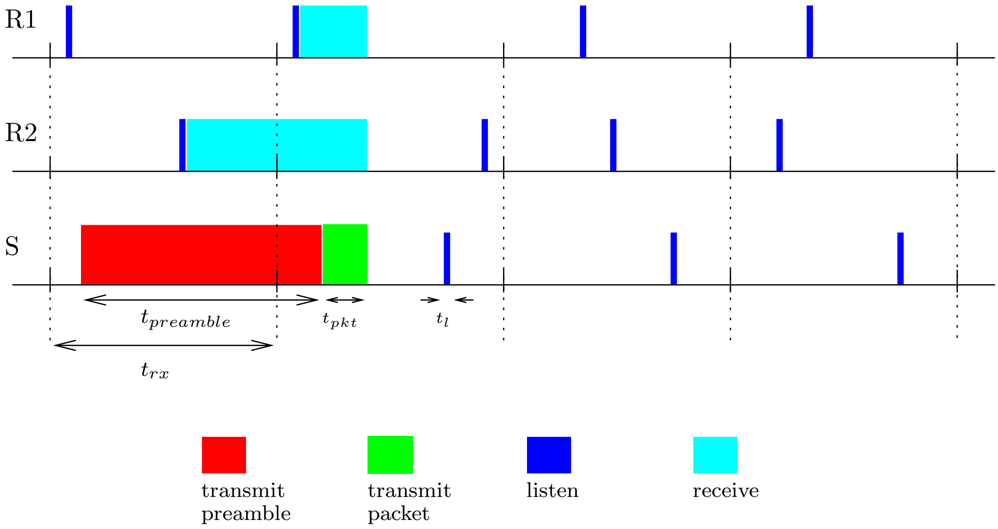


Figure 2.3: Illustration of asynchronous duty cycling. Each node wakes up for a short interval t_l, with wakeup period t_{rx}. The sender does not know when in the interval t_{rx} the destination will next wake up; it must therefore send a long *wakeup preamble* (of length $t_{preamble} > t_{rx}$) before its packet.

(receiver 2), or they may be periodic (receiver 1); in either case the phase may be shifted after a packet reception, as is the case for receiver 1. The key point is that nodes are not synchronized, so from the sender's point of view there is little difference between a deterministic or random wakeup schedule.

Using this long preamble is the simplest possible approach to communicate with a node whose listening schedule is unknown. Other more complex schemes are possible. For example, repeatedly sending the packet for the duration t_{rx}, or embedding in the preamble a countdown sequence indicating how many preamble bytes are left until the packet itself. Note that while such optimizations may allow for slightly more efficient implementations of LPL, they do not fundamentally change the fact that nodes must send transmissions of length t_{rx}.

WiseMAC: from individual to asynchronous duty cycling

WiseMAC [25] is an elegant form of duty cycling that espouses elements both of individual (synchronous) duty cycling, and asynchronous duty cycling. Consider a node i having a packet for a destination j. If i has already learned j's schedule, it waits until j's next wakeup time to transmit its packet, thus operating similar to individual duty cycling. Otherwise, i can simply transmit a packet with a long preamble, as in asynchronous duty cycling, since i knows that j will wakeup at some point in an interval of length t_{rx}. Node j indicates its next wakeup time to node i by piggy-backing it in its acknowledgement frame, and i now has learned the schedule of this at the cost of a long preamble transmission.

WiseMAC enables a smooth transition from synchronous to asynchronous duty cycling by adjusting the length of the preamble to the level of uncertainty about a receiver's next wakeup time. Specifically, if the next packet from i to j comes long after the previous one, clock drifts will mean that synchronization between the two nodes has loosened. To account for this clock drift, the preamble is extended by a length that is a function of the elapsed time since the last message exchange. The overall effect of these measures is that WiseMAC adapts automatically to traffic fluctuations. Under low load, WiseMAC uses long preambles and minimizes receiver costs (idle listening); under high loads, WiseMAC uses short preambles and minimizes transmitter costs (wakeup preambles).

2.4.4 Comparison of Duty Cycling Schemes

Each of the class of duty cycling schemes presented above has its own advantages and drawbacks, which we outline here. Our explicit goal is *not* to promote one type of scheme as the most effective form of duty cycling. Both the research community and standardization bodies are still actively investigating the design and performance of duty cycling schemes. Furthermore, this is a young area of research and it may turn out that entirely different techniques turn out to be best[1]. Our aim is to show that the contributions of this thesis are general and go beyond optimizing one specific duty cycling scheme (in fact, anypath routing is relevant also beyond duty-cycled wireless networks).

Synchronous duty cyling

The key advantage of synchronization is that nodes know precisely when to wake up and (if necessary) transmit, thus avoiding the need for long wake up preambles. If synchronization were easy to achieve, the debate would most likely be over, and asynchronous schemes would not be considered. Unfortunately this is not the case, and the drawbacks of synchronous duty cycling come directly from this difficulty. The first is that synchronization always requires some form of communication. In an ideal world, a single message exchange would suffice to align nodes forever, but in practice, clock drifts mean that synchronization must be periodically refreshed, with the refresh

---

[1] A particular favorite of ours are synchronization schemes based on concepts of loosely coupled oscillators [106]; we speculate that these can and will be successfully applied to duty cycling in the future.

period being inversely proportional to the degree of synchronization preciseness that is required. Furthermore, synchronized schemes are less robust and error-tolerant than asynchronous schemes, since they require nodes to track and maintain schedule state; if this state is lost or corrupted, nodes must go through a costly re-acquisition process.

Clustered duty cycling

We now discuss characteristics that are specific to each particular flavor of synchronous duty cycling. Clustered schemes such as S-MAC or T-MAC, have the advantage that nodes are all either asleep or awake at the same time. During the active period, transmissions can occur just as they would in an always-on network; this allows a clean separation between the duty cycling mechanism and the actual transmission and reception of messages. The key drawback is that it requires convergence to a common schedule in order to be effective; as mentioned earlier this convergence is hard to reach and imposes protocol overhead. In a worst-case scenario where each node diverges to its own schedule, nodes will have to listen during each neighbor's active period, resulting in a duty cycle of $|N(i)|t_l/t_{rx}$. Note that this degenerate case is *not* equivalent to individual duty cycling, and is in fact significantly worse, because a node must wake up for each neighbor's active period. In other words, the worst-case number of schedules in the neighborhood of a node i goes to $|N(i)|^2$ with clustered duty cycling, whereas it is $|N(i)|$ with individual duty cycling.

Individual duty cycling

Individual duty cycling has the same overall merits and drawbacks of synchronous schemes that we have outlined above. In comparison with clustered duty cycling, individual duty cycling has no shared broadcast channel, since there is no common wakeup time. This is a mixed blessing. On the one hand, nodes do not have to overhear packets destined to other nodes, and so they spend less time listening without reason. On the other hand, sending a broadcast packet has higher cost than a unicast packet, since it must either be sent to each neighbor individually, or with a very long preamble that covers each neighbor's wakeup times.

A final merit is that for a fixed duty cycle t_l/t_{rx}, individual duty cycling allows higher channel utilization than clustered duty cycling. This can be seen by observing that within a neighborhood, node i's wakeup interval of duration t_l is only shared by any nodes in $N(i)$ that have a message to transmit to i. In clustered duty cycling, this same interval t_l must be shared by *all* possible pairwise communications between nodes in the neighborhood. We do not expect this aspect to be critical given our focus on low-rate networks, but it is nonetheless worth pointing out, especially for networks with highly bursty traffic.

Asynchronous duty cycling

The major advantage of asynchronous duty cycling is that it requires no explicit coordination between nodes. Therefore there is no background traffic of any sort to communicate schedules or maintain synchronization. Furthermore, being stateless, this approach has no robustness issues or failure modes that can arise from losing synchronization.

This robustness and simplicity are likely explanations for the fact that asynchronous duty cycling has been frequently adopted in experimental low-power wireless systems [90,97,98]. The other advantage is that it minimizes the overhead of listening time when there is very little or no traffic. However, asynchronous duty cycling has two important drawbacks. First, long preambles significantly increase the burden on transmitters. Since decreasing the duty cycle requires increasing t_{rx}, the transmission cost is inversely proportional to the duty cycle[2]. This constitutes an obstacle to running nodes at duty cycles below a few percent, since the sending cost then becomes prohibitive. Second, selecting an optimal value for the parameters (t_l and t_{rx}) requires periodic traffic as well as knowledge of traffic rate, network size, and density.

2.5 Packet Combining

To the best of our knowledge, the only existing work to have analyzed, designed, and implemented a working system based on multi-point packet combining is the Multi-Radio Diversity (MRD) system of Miu et. al. [76]. MRD exploits the spatial diversity generated by having multiple receiving base-stations placed at distributed locations, and effectively turns these base-stations into a distributed antenna array. MRD employs a novel block-based merging algorithm that is well-suited to bursty error characteristics. Beyond the different physical layer technologies employed, the fundamental (and complementary) difference between MRD and the scheme proposed here is that MRD works only for single-hop interactions, with multiple radios receiving from one sender, whereas our scheme considers a single radio receiving from multiple (or single) senders.

A large and fast-growing body of work in information and communication theory

---

[2] Decreasing t_l beyond some (radio-dependent) minimum value is not an option to reduce the duty cycle, because nodes must wakeup for a minimum duration in order to stabilize the radio hardware and listen for a few bytes' worth of time.

addresses cooperative diversity in various settings. While a survey of this burgeoning field is beyond the scope of this dissertation, we should mention the recent work of Laneman, Tse and Wornell [59], that compares the outage behavior of different relaying schemes (amplify-and-forward, decode-and-forward, selective relaying) in a 4-node network.

Our work is closest in spirit to that of Valenti and Zhao [113] who give a generalization of Hybrid-ARQ in the context of a diversity routing protocol. They provide analytical and numerical results on the outage behavior and throughput gains achievable with hybrid-ARQ. Note that their work is theoretical in nature, assuming for example the use of optimal (in an information-theoretic sense) channel codes with no constraints on block length or decoding complexity.

To our knowledge, the only existing work examining packet combining in the context of practical sensor networks is that of Koepke [57]. This early work examines through simulation the gains of packet combining in a single-hop setting, using only repetition coding.

While their focus is not directly on packet combining, Zhao and Govindan [114] present detailed measurements of packet delivery performance at the physical and MAC layers. At the physical layer, they examine the difference between three static channel codes. They observe that the most powerful code (SECDED) best alleviates the variability of “gray” links; however it also imposes the highest redundancy on reliable links. This observation is in line with the intuition that no static code can be well-matched to the large diversity of links in a wireless network.

2.6 Summary

This chapter has provided an overview of related work and contrasted it with the contributions of this dissertation. We have illustrated how existing forms of multipath routing are driven by different motivations and use other mechanisms than anypath routing, despite superficial similarities.

We described existing work in anycast forwarding and anypath routing, distinguishing between schemes that use geographic positions, schemes based on single-path routing, and schemes based on an existing multi-path routing protocol. We further explained why neither of these schemes is entirely satisfactory.

Finally, we surveyed existing ultra-low power link-layers for unicast forwarding, that form the basis of the low-power anycast forwarding models of Chapter 4, and of the system designed in Chapter 7.

Chapter 3

Network Metrics and Models

> *All models are wrong. Some models are useful.*
> - George Box

Anypath routing is a general technique that works over many classes of networks. Even within the class of wireless networks, there are many different sub-classes, and one way to distinguish different families of wireless networks is by comparing their link and physical layers, and the *costs* that arise from using these link and physical layers.

The goal of this chapter is thus to introduce some wireless network models and set the stage for the development of anypath routing. Along the way, the exposition shall show how this dissertation builds on prior research in wireless link layer.

The chosen models do not seek to provide an exhaustive taxonomy of all possible wireless link layers; nor does each model attempt to capture every detail with excruciating precision. Rather, our aim is to provide a small number of simple models that capture the salient features of wireless links; these models should also allow us to reason cleanly and with a good level of abstraction about protocol design choices.

With each network model we associate a *cost metric*. Again, our cost metrics are simple and aim to characterize the most critical performance aspect of each given model, rather than incorporating every cost component in minute detail.

The chosen cost metrics and link models are summarized below. The first two are primarily used in conjunction with *always-on* links, and the remaining ones are used in conjunction with *duty-cycled* links, where radio transceivers are powered down as often as possible to preserve energy.

- The **end-to-end** (E2E) delivery probability cost metric captures the probability of a packet being reliably received over a given link or route. This metric is used for links that have no reliability mechanism.

- The **expected transmission count** (ETX) is the average number of transmissions required to reliably send a packet across a link or route. Selecting routes with low numbers of transmissions allows to increase throughput as well as to save energy.

- The **transmission energy** cost is used in conjunction with asynchronous duty-cycled links, where nodes awaken at random times and sample the channel for activity. This metric reflects the energy that is required to transmit to a node that may be sleeping. Minimizing this cost allows to increase the energy autonomy of a network.

- The **latency** cost metric counts the expected delay to transmit a packet across a link or route, and is used in conjunction with synchronous duty-cycled links, where nodes coordinate their sleep and wakeup times.

3.1 Overview and Notations

We model a network as a directed weighted graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is a finite set of nodes, and $\mathcal{A}$ is a set of $|\mathcal{A}|$ pairs of distinct nodes (i, j) from $\mathcal{N}$ between which a link exists.

Each link $(i, j) \in \mathcal{A}$ also has an associated distance d_{ij}. The choice of distances is an important modelling decision that should reflect the underlying cost of communicating across each link, and we shall see in this chapter that there are many ways to assign distances to links. We shall discuss link distances in Section 3.4. One assumption that shall hold throughout this dissertation is that link distances are always positive. The simplest distance assignment is of course to set $d_{ij} = 1$ for every link (i, j) in $\mathcal{A}$. Independently of the assignment of distances to links in $\mathcal{A}$, we use the convention that $d_{ij} = \infty$ if (i, j) is not an link of the graph, and $d_{ii} = 0$. Any node j for which $d_{ij} < \infty$ is a *neighbor* of node i, and the set of neighbors of i is denoted $N(i)$. The *average degree* (or density) ρ of a network is the average number of neighbors per node, i.e., $\rho = |\mathcal{N}|^{-1} \sum_{i \in \mathcal{N}} |N(i)|$.

A *path* is a sequence of nodes $(1, 2, \ldots, n)$ such that each of the pairs $(1, 2), (2, 3), \ldots, (l-1, l)$ are arcs of $\mathcal{G}$. Given any path $(i, j, k, l \ldots, m)$, the length is defined as $d_{ij} + d_{jk} + \ldots d_{lm}$. We will equivalently refer to path length or path cost, and to link distance or link cost, depending on the context.

Each link $(i, j) \in \mathcal{A}$ has an associated packet delivery probability p_{ij}. This probability is a characterization of the physical channel between the two nodes i and j. It is therefore an inherent property of each network and, unlike the assignment of distances d_{ij}, the assignment of p_{ij} is given and cannot be modified by the protocol designer as part of an overall system model. We shall discuss link probabilities in Section 3.3.1.

The cost and packet delivery probability of a link are usually related in some way that depends on the specific network model. For example, one might define $d_{ij} = -\log p_{ij}$, in which case link and path costs are determined exclusively by their packet delivery probabilities. Or, one might define d_{ij} to be the energy cost for i to transmit a packet to j, whether or not that packet is successfully received.

This model does not attempt to cover networks that evolve over time, due for example to growth or mobility processes; we do however explore time-varying networks empirically in Chapter 6.

As a final point of notation, if X is a set, then 2^X stands for the power set of all $2^{|X|}$ subsets of X.

3.2 Spatial and Temporal Traffic Patterns

Traffic patterns play an important role in the overall performance of networked protocols. Formulating a complete statistical description of data traffic in any network is difficult. For the purpose of this dissertation, it is sufficient to describe traffic patterns at a fairly coarse level. We first distinguish between *spatial* and *temporal* aspects of traffic.

The spatial distribution of traffic describes which node pairs are most likely to communicate. This dissertation focuses on the *unicast* service model, where each packet is sent to one destination. We do not consider multicast or broadcast traffic, where packets are sent to a group of nodes or to the entire network. With unicast traffic, different spatial patterns are possible. For example, one could have a traffic pattern where all node pairs are likely to communicate. A more restricted traffic pattern is *many-to-one*, where all nodes send packets to the same destination. This traffic pattern is representative of many sensor network applications, where nodes monitor some parameter of their physical environment and report to the base station. In this context, many-to-one traffic is often referred to as *data gathering* [111]. The contributions of this dissertation are equally applicable to all spatial traffic patterns; however, our protocol design and testbed evaluations of Chapter 7 focuses on wireless sensor networks and consequently uses data gathering traffic patterns.

The temporal distribution of traffic describes how packets in a given source-destination pair are distributed in time. Temporal traffic models can be extremely simple (for example, each node generates packets periodically) or rather more complex (e.g., self-similar traffic [63]). We do not make any specific assumptions about such types of temporal traffic behavior.

Our only modelling assumption is that *the network is not consistently saturated with traffic*. Informally, this means that while collisions and congestion may occur, they are not the overriding issue when it comes to achieving high network performance. We shall study the primary performance aspects of a network in a non-saturated regime, which are reliability, latency, and energy efficiency.

3.3 Link Layer Aspects

There are a large number of possible modelling and design choices to represent a wireless link, varying both in complexity and in realism. We describe here the link layer modelling and design space that is explored in this dissertation. We do not use a single monolithic model. Rather, we propose a number of key modelling building blocks that can be assembled to obtain more or less complex representations of a wireless network.

The first choice (Sect 3.3.1) concerns the modelling of channels as reliable or unreliable, and the second (Section 3.3.3) is whether or not link-layer retransmissions are employed. In the case of low-power links, we must also choose a model of duty cycling (according to the schemes described Section 2.4), or else assume that links are permanently on and do not use duty cycling.

3.3.1 Reliability and Error Models

At the physical layer, the wireless media provides a broadcast channel between a node and all neighbors that are within transmission range. The notion of transmission range depends on parameters of the physical layer such as modulation, bit-rate, channel bandwidth, and transmission power. In this dissertation, we shall consider the above parameters to be either fixed, or to change only at timescales that are significantly larger than the timescales over which a network-wide routing protocol converges. We therefore exclude schemes that dynamically change these physical layer parameters, for example adaptive *power control* schemes, or adaptive *channel coding* schemes. This exclusion is not due to any fundamental incompatibility between the schemes proposed in this dissertation and adaptive physical layers, but rather to keep a model of underlying physical layer that is reasonably simple and allows us to study in clean isolation the performance of the mechanisms that we develop above. We are also motivated in this choice by the fact that adaptive physical layer schemes are difficult to make work in practice in multi-hop wireless settings, and (to the best of our knowledge) no such scheme has successfully been applied in real-world multi-hop wireless systems.

In practice, manufacturers of radio transceivers usually provide a specification of the transceiver's *receiver sensitivity*, typically defined as the minimum signal to interference noise ratio (SINR) at which the transceiver can decode received bits reliably (where reliably means with a very high probability, such as 0.9999). A detailed discussion of physical layers is beyond the scope of this dissertation, and we refer to [33, 101] for a more in-depth survey.

We use two simple models of link errors and reliability in this dissertation: *unreliable links* and *reliable links*. With the unreliable link model, we consider links as unreliable packet channels: a packet is received successfully with a given probability, which may depend on the physical distance between two nodes, or on obstacles and multi-path fading effects. The second model is motivated by the use of link filtering (described below) in many real-world protocols. It considers a binary form of connectivity: either two nodes can communicate reliably, or they cannot communicate at all.

Unreliable links

The unreliable link model captures the unreliability of such wireless channels in a simple manner: a packet sent from node i is received by node j with probability p_{ij}. We will call p_{ij} the *packet delivery probability* between i and j. We make two assumptions of independence concerning packet delivery: *spatial* independence, and *temporal* independence. By spatial independence, we assume that reception at different nodes is independent. Specifically, a packet sent by a node i is received independently, with probability p_{ij}, by each of the neighbors $j \in N(i)$. Spatial independence is a valid assumption if nodes are sufficiently far apart, so that the fading coefficients are not correlated, and if interference is not an important source of packet loss. By temporal independence, we assume that successive packet receptions between two nodes are independent. For this assumption to be valid again requires that interference is not the dominant source of packet loss, and that channel characteristics are stationary.

In a real wireless network, the link delivery probabilities p_{ij} change over time, due to environmental changes, external sources of interference, or the physical movement of objects that change how the radio signal is affected by multi-path shadowing. We do not seek to capture these effects with this model.

Note that one may also consider transmissions at a finer granularity and use the bit-error probability, that is the probability of each bit in a packet being received in error. However going from the bit-error probability to the packet delivery probability requires assumptions on the joint bit error statistics (e.g., bit errors may be independent or bursty), on packet lengths, and on the properties of error detection and correction mechanisms that may be used at the physical layer. While packet error probability is a coarser-grained approximation of a link behavior, it allows us to consider communication between nodes without entering the specifics of bit-error distributions, and at a practical level, link and network-layer protocols can estimate packet delivery probability far more easily than more fine-grained statistics on bit-error distributions.

Reliable links

Most routing protocols [9, 17, 29, 108] for wireless networks maintain *estimates* of the link quality p_{ij} between neighboring nodes. These estimates can be made, for example, with active measurements: nodes send periodic beacon messages and keep track of gaps in sequence numbers to detect losses. Or, if there is a known minimum traffic rate, and if this rate is sufficiently high, passive measurements of packet loss may be possible using only data packets.

At the same time, many experiences [13, 32, 114] with live wireless networks have shown that links can be characterized roughly as persistently poor, intermediate, or persistently good, and that the intermediate links tend to fluctuate much more than the good and bad links. The intuitive explanation is that an intermediate link is very close to the sensitivity threshold, and very slight changes in the radio frequency (RF) environment (such as interference level, path attenuation, or short-term fading) are enough to move this link to either side of the sensitivity threshold. Given the high variability of intermediate links, estimates of their quality will typically be far less accurate than estimates of “very good” or “very bad” links.

Therefore, it is common for routing protocols to perform *link filtering*, whereby all links with delivery estimates below a certain threshold (typically placed in the range 0.8-0.9) are filtered and not taken into consideration by the routing protocol. An additional reason why many protocols do such filtering is also that it simplifies the design of the routing subsystem sitting above the filtering function: once filtering characterizes links as either “up” or “down”, it becomes possible to use simple distance metrics such as hop-counts to drive routing protocols.

Based on these protocols, we define the *reliable links* model as a network in which all p_{ij}’s are either 0 or 1. Note of course that this an intentionally simple characterization, and that in reality even those links which remain after filtering are not completely reliable (given our typical filtering threshold of 0.8-0.9).

3.3.2 Media Access and Carrier Sensing

When nodes use a shared medium, such as a wireless medium, it is important to have a mechanism in place to avoid that two neighboring nodes transmit at the same time. Otherwise, collisions may occur, and colliding packets may end up being lost at their intended receiver(s). The protocol building block that implements this function is usually known as the Media Access Control (MAC), and sits right above the physical layer. MAC mechanisms can be broadly separated into two categories: scheduled access and random access.

In the first category, a schedule establishes when and for how long each node may transmit on the channel. The most common form of scheduled access is Time-Division Multiple Access (TDMA), in which the channel is divided into slots, which are assigned to individual nodes. TDMA is used in some cell-based systems (e.g., GSM [77]), where it is possible for a base-station to assign precise schedules to mobile terminals.

In the second category, random access, nodes do not have preassigned schedules, and they attempt to access the channel on an on-demand basis. The Carrier Sense Multiple Access (CSMA) protocol, originally introduced by Kleinrock and Tobagi in 1975, tries to do just that. CSMA is a probabilistic media access protocol (MAC) where a node listens to the wireless channel before sending. That is, it tries to detect the presence of an encoded signal from another node before attempting to transmit. If a transmission is ongoing, the node waits for the transmission in progress to finish before initiating its own transmission. More precisely, it waits for some *back-off* interval which is randomly selected and should be at least longer than the ongoing transmission time. Different random backoff strategies are possible, a well-known one being binary exponential backoff [34].

Another popular class of schemes employs virtual carrier sensing, originally proposed by Karn with the MACA protocol [49] and subsequently refined by Bhargavan et al with MACAW [8]. With virtual carrier sensing, nodes use a three-way handshake to avoid collisions at common neighbors (so called “hidden terminal effect”). The sender initiates the handshake by transmitting a short Request-To-Send (RTS) control packet announcing its desire to use the channel. The receiver responds with a Clear-To-Send (CTS) packet, after which the channel is cleared for the sender to use. Note that virtual carrier sensing can be layered over CSMA, meaning that a sender can employ CSMA before transmitting its RTS packet. The advantage that virtual carrier sensing brings is that it avoids the *hidden terminal* problem, which happens when a “hidden” node is sending a packet to the destination which the sender cannot overhear. This hidden terminal effect can potentially happen whenever not all nodes are within range of each other. Of course virtual carrier sensing also comes with some overhead, namely the time and bandwidth spent for the extra RTS/CTS signalling. Whether or not this overhead is worthwhile depends essentially on parameters such as traffic patterns and packet sizes. Virtual carrier sensing is used in the IEEE 802.11 wireless LAN standard [78], where the overhead of 20-30 byte signalling packets is reasonable in comparison to frames of 1500 bytes. The case for using virtual carrier sensing is less clear in sensor networks, where packet sizes are extremely small. The 802.15.4 standard for example does not employ virtual carrier sensing.

3.3.3 Retransmissions

In order to increase reliability, wireless link layers often include a retransmission scheme: after transmitting a unicast packet to a neighbor, erroneous or lost packets between a node and its neighbor are retransmitted, using what is known as Automatic Repeat reQuest, or ARQ [7]. ARQ requires a feedback channel from the receiver to the sender, that in practice is implemented with short frames that are sent from the receiver to the sender.

The simplest ARQ protocol is called *stop-and-wait*. Here the aim of the protocol is to ensure that a packet has been received properly before transmitting the next packet. The receiver transmits a positive acknowledgement (ACK) to the sender each time it receives a unicast packet (addressed to itself). The transmitting node awaits an ACK for some period, and retransmits the packet if it has not received the ACK at the end of this period.

More sophisticated ARQ schemes are possible. If packets are marked with increasing sequence numbers, a receiver can detect gaps in the stream and request selective retransmissions by sending selective acknowledgements containing the sequence number(s) of the packets that have not been received. The chief advantage of these schemes over stop-and-wait is that they allow having multiple outstanding packets, and thus can attain a higher link throughput. This advantage comes at the cost of some added complexity and memory requirements.

In the remainder of this dissertation, we shall only consider stop-and-wait ARQ, because throughput gains from other ARQ schemes are not apparent with low traffic rates, and because of its analytical simplicity and low resource usage for our implementation. We do note however that none of the schemes presented in this dissertation fundamentally presuppose stop-and-wait ARQ.

Duty Cycling	Retransmissions	Channel
None	Yes	Unreliable
None	No	Unreliable
Clustered	Yes	Unreliable
Individual	No	Reliable (or filtered)
Asynchronous	Yes	Reliable (or filtered)

Table 3.1: Link-layer models considered in this dissertation. According to the criteria of duty-cycling, retransmissions, and channel reliability, a total of 16 combinations are possible; the selected subset amounts to a representative segment of all possible wireless link layers.

3.3.4 Link Layer Models

In this we have introduced two key choices that must be made to model a wireless link layer. We have not listed here the choice between different forms of media access and carrier sensing; we shall assume throughout this dissertation that traditional CSMA is employed, but the schemes presented here can be carried over to systems that use virtual carrier sensing.

While not all combinations make sense (i.e., with reliable links we would not need ARQ), we still have over 10 different models to choose from if we attempt to cover all possible combinations. We will consider a subset of 5 models. The first two models are considered for their generality, and because of their relevance to clustered duty cycling schemes. The other three models are considered because taken together they capture the key ideas behind a large portion of the many link layer schemes proposed for low-power wireless networks (we refer to [4, 60] for good surveys of this area).

The first model simply considers "bare" unreliable links (without retransmissions). While in practice few wireless link layers operate without any mechanism to counteract channel unreliability, this model is interesting due to its extreme simplicity, and it allows us to examine the reliability of paths chosen by a routing protocol. The second adds ARQ retransmissions for reliability, and leads to a link cost metric that is frequently used in wireless routing protocols [17].

The next three models all incorporate some form of duty cycling. We shall consider both forms of synchronized duty-cycling (clustered and individual), as well as asynchronous duty cycling, with filtered links assumed for the latter two. Assuming filtered links is a modelling decision that we make in order to focus on specific metrics of interest to these duty-cycled (energy and latency) schemes and for analytical tractability. The experimental portion of this dissertation (Chapter 7) shall serve to verify if these modelling decisions are reasonable or not. Table 3.1 summarizes the above list. As a final point, note that the use of retransmissions with reliable links (asynchronous duty-cycling) is not contradictory, because even with error-free channels, unreliability can be introduced if we send packets with a preamble that is shorter than t_{rx}, as we shall intentionally do in the next chapter.

3.4 Link and Path Cost Metrics

The link costs d_{ij} can be assigned to links in a number of different ways. Unlike the link probabilities p_{ij}, that are an internal property of each given network and cannot be controlled, the assignment of costs (or distances) to links is both a modelling and design choice. Which type of distance is appropriate depends on the cost criterion and problem that one is interested in. Intuitively, we simply would like d_{ij} to reflect, in some manner, the cost for node i to transmit to node j. The only constraints on our choice of link costs are that they can be *added* to obtain cumulative path costs, in accordance with our earlier definition of path length, and that they are strictly positive.

This section defines a number of possible link costs and associated path costs, that we will use to design and evaluate the protocols proposed in this dissertation. We distinguish four classes of cost metrics, namely:

- Metrics based on *delivery probabilities*. We use these metrics when we are concerned with the delivery of packets across a link, and with the overall reliability of a path.

- Metrics based on *(re-)transmission counts*. With these metrics, we are concerned with the number of times a packet must be retransmitted to get across a link or path. In other words, these metrics model the cost, in number of transmissions, to turn an unreliable link (or path) into a reliable one.

- Metrics based on *latency*. With these metrics, we count the delay to transmit a packet across a link or path.

- Metrics based on *energy costs*. These seek to measure the cost in energy to transmit a packet along a link or path. Since we have assumed the use of nodes with fixed transmission power, the duration of a packet transmission is a good approximation of energy expense.

In principle, a cost metric from each of these classes can be defined for every link layer model seen in Section 3.3.4. However, rather than consider each class of metric for each link model, we shall choose a single type link metric in conjunction with each model, selecting the one which is most relevant. For example, a transmission-count metric is of little use for a link without ARQ. Also, a delivery probability metric is irrelevant with reliable links, and for unreliable links with retransmissions, since we are assuming that a packet is retransmitted until it is correctly received[1].

To lighten the notation of this section, we shall not repeatedly state that (i, j) is a link of $\mathcal{A}$ each time we refer to d_{ij} in this section. The convention that $d_{ij} = \infty$ if $(i, j) \notin \mathcal{A}$ and $d_{ii} = 0$ remain valid for any choice of link cost.

3.4.1 Always-On Link Costs

Delivery probability cost metrics

With unreliable links, a natural path cost metric is the end-to-end delivery probability (E2E). This cost metric can be obtained from the raw link delivery probabilities by taking the negative logarithms of these probabilities.

Definition 3.1 (E2E metric)**.** *For unreliable links without retransmissions, we use the* end-to-end delivery probability *(E2E) metric, defined as:*

$$d_{ij} = -\log p_{ij}.$$

This metric represents the end-to-end delivery probability of a packet along a link or path. If, as we have assumed, links are independent, then finding the shortest path between two nodes n_1 and n_l using distances $-\log p_{ij}$ is equivalent to finding the most reliable path between those two nodes. In fact, the probability p of packet delivery along a path $(n_1, \ldots, n_l)$ from n_1 to n_l can then be computed from the path's cost D:

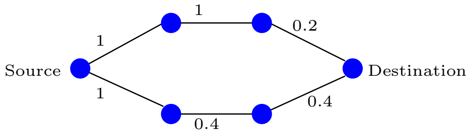
$$\begin{aligned} e^{-\log D} &= e^{-\log p_{12} - \log p_{23} \ldots - \log p_{l\text{-}1\,l}} \\ &= e^{-\log(p_{12}p_{23}\ldots p_{l\text{-}1\,l})} \\ &= p_{12}p_{23}\ldots p_{l\text{-}1\,l} \\ &= p. \end{aligned}$$

Transmission-count cost metrics

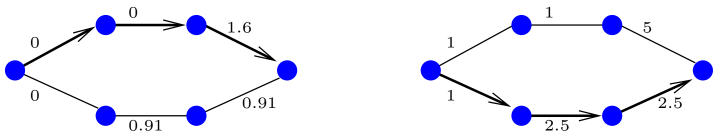
With link-layer retransmissions, the cost of a link is naturally modelled by the *expected transmission count* (ETX) [17], which is the expected number of packet transmissions needed by node i to send a packet to neighbor j. The expected transmission count can be defined in different ways, depending essentially on how the feedback channel

---

[1] Of course, this model constitutes an idealization, and in practice wireless links remain unreliable even with link-layer retransmissions.



(a) Delivery probabilities p_{ij}



(b) E2E: $d_{ij} = -\log p_{ij}$ (c) ETX: $d_{ij} = \frac{1}{p_{ij}}$

Figure 3.1: Comparison of shortest path routing under end-to-end reliability (E2E) and expected transmission count (ETX). Under E2E, the shortest path goes through the upper two nodes, and under ETX the shortest path goes through the lower two nodes. Illustration of Example 3.1.

from j to i is modelled. We will consider two cases here. In the first, we assume that acknowledgement frames are reliably received by node i, and so i never makes any spurious retransmissions due to missing an acknowledgement. In this case, the expected transmission count is $d_{ij} = 1/p_{ij}$. In the second case, we take into account the possibility that the acknowledgement from j to i can be lost, and the expected transmission cost is then $d_{ij} = 1/(p_{ji}p_{ij})$, where here we have assumed that the probability of reception is the same for ACK frames as for data packets.

Definition 3.2 (ETX metric)**.** *For unreliable links with retransmissions, we use the expected transmission count metric (ETX), defined as:*

$$d_{ij} = \frac{1}{p_{ij}}$$

Example 3.1 (Different shortest paths under E2E and ETX)**.** While both the E2E and ETX metrics are solely based on underlying link delivery probabilities, observe that they do not give rise to the same shortest paths. A simple example is given in Figure 3.1. Note also that while the expected transmission count for a *link* can be computed from the delivery probability of that link, and vice-versa, this one-to-one correspondence does *not* carry over to path costs.

3.4.2 Duty-Cycled Link Costs

The motivation for employing duty cycling is to increase energy efficiency, and so it is natural in this context to seek metrics that reflect the *energy cost* of using a link or path. Another aspect of interest is *latency*, because of the inherent tradeoff between energy efficiency and latency. In particular, duty cycling inevitably introduces additional latency across each link, either from having to wait until the next active period (synchronous duty-cycling), or because of the long preamble that must precede a packet (asynchronous).

For each duty-cycling scheme, there is a tradeoff between decreasing the duty-cycle t_l/t_{rx} and increasing some other form of cost. In essence, decreasing the duty cycle means increasing t_{rx}, which either increases latency (in synchronous duty cycling), or increases transmission cost (by requiring a longer preamble in asynchronous schemes). Of course decreasing the duty-cycle can also to some extent be achieved by decreasing t_l, but nodes must in practice wake up for a minimum duration which includes the time to detect a possible ongoing transmission, and the settling time of analog circuitry. We shall therefore seek metrics that in some way expose the tradeoff against decreasing duty cycles.

Synchronous duty cycling

With synchronized duty cycling, the energy cost to transmit a single packet is independent of the duty-cycle, since the packet frame and duration are fixed (unlike for asynchronous duty cycling). What does change is the latency.

In fact, with synchronous duty cycling, latency increases as the inverse of the duty cycle, and for this reason we use latency as our primary metric for synchronous duty cycling. Recall that each node wakes up with a fixed period of t_{rx}, and assume that the phases of each node's schedule are random and uniformly distributed in $[0, t_{rx}]$. Consider a node that needs to send a packet to a neighbor. Either we are inside the destination's active period (of duration t_l), in which case the packet can be immediately sent, or we are outside, in which case we must wait until the next active period. The expected delay until the packet can be transmitted is therefore

$$0 \cdot \frac{t_l}{t_{rx}} + (1 - \frac{t_l}{t_{rx}})\frac{t_{rx}}{2}.$$

While this expression of latency is exact for individual duty cycling, it is not entirely accurate for the case of clustered duty cycling. With clustered duty cycling, if the packet that our node needs to send was received from a neighbor and is being forwarded by the routing protocol, then the latency will often be lower than the value given above, because a node having just received a packet is likely to be still in the active period. Modelling of this effect would require arbitrary assumptions on network topology, routes, and traffic patterns, which we do not consider here. As we shall see in the following chapter, the fact that this expression is an approximation for the case of clustered duty cycling is not of great concern, because latency of clustered duty cycling is unchanged by anypath routing, and so a precise comparison is not required.

Returning to the expression of latency above, we note that it is further simplified for low duty cycles, i.e., when $t_l \ll t_{rx}$. Since we are primarily interested in duty cycles of 1% or lower, we adopt this as our definition of latency cost for synchronous duty cycling:

Definition 3.3 (Latency metric)**.** *For links with synchronized duty-cycling, we use the* latency *metric, defined as:*

$$d_{ij} = \frac{t_{rx}}{2}$$

Of course, choosing latency as our cost metric is a modelling *choice*, and we could also have chosen to use a metric based on expected transmission count, delivery probability, transmission energy cost, or even a metric combining these aspects. However, these metrics remain unchanged as we decrease the duty cycle, and for low-power networks we are interested in seeing what happens as we reduce the duty cycle towards extremely low values.

Asynchronous duty cycling

Like in the synchronous case, latency increases as we reduce the duty cycle, because the expected duration until our neighbor next wakes up is proportional to t_{rx}. We might therefore define a latency metric similar to the one for synchronous duty cycling. However, there is an additional tradeoff with decreasing the duty cycle, which is that the *energy cost* of a transmission also increases, because our preamble length must be as long as t_{rx}.

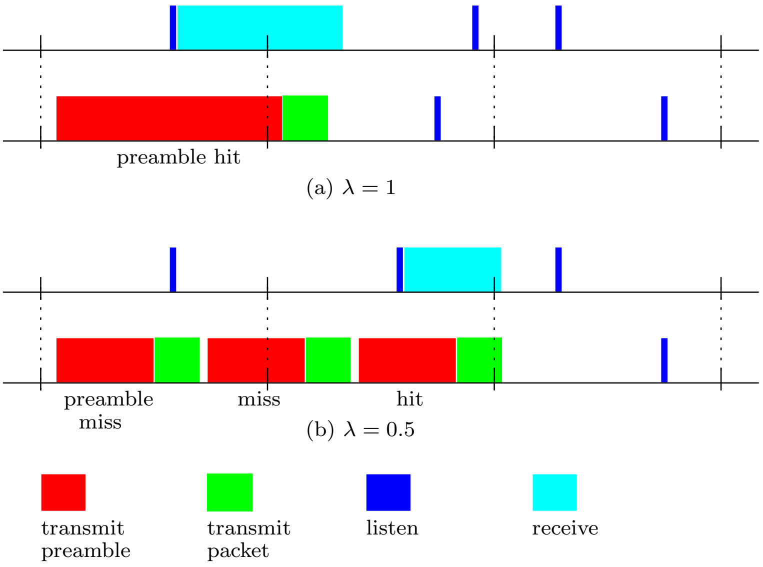
Given that the prime goal of duty cycling is to preserve energy, this tradeoff between transmission energy cost and duty cycle is our prime concern and we therefore choose to define an energy metric for asynchronous duty cycling. Noting $t_{preamble}$ to be the duration of the preamble sent by the transmitter and t_{pkt} to be the transmission time of a packet, we can define our energy cost metric as:

Definition 3.4 (Energy transmission cost metric)**.** *For asynchronous duty-cycled links, we use the* energy-cost *metric, defined as:*

$$d_{ij} = t_{preamble} + t_{pkt}$$

Reducing preamble length is suboptimal

Let us consider a naive strategy to reduce the cost of sending wakeup preambles with asynchronous duty cycling. Define a *preamble hit* (resp. *preamble miss*) as the



(a) $\lambda = 1$

(b) $\lambda = 0.5$

Figure 3.2: Transmission cost increases with reduced length preambles. In the upper diagram, the sender uses $\lambda = 1$ and hits the receiver after one transmission. In the lower diagram, the sender uses shorter preambles but must make three transmissions until the first hit.

event happening when a node awakens (resp. does not awaken) during the preamble transmission of a packet that is destined to this node. Note that a preamble miss cannot occur as long as $t_{preamble} \geq t_{rx}$[2]. The naive strategy consists of sending preambles of reduced duration ($t_{preamble} < t_{rx}$), and using retransmissions when a packet is not received by the next hop due to a preamble miss. If we use a preamble of length $t_{preamble} = \lambda t_{rx}$ ($0 < \lambda < 1$), and assuming that the wakeup time is uniformly distributed inside the interval t_{rx}, the probability of preamble hit is $p_{hit} = \lambda$. However, a simple computation shows that this strategy is not effective: the expected transmission count becomes $1/\lambda$, and the expected cost of sending a packet is:

$$d_{ij} = \frac{t_{preamble} + t_{pkt}}{\lambda} = t_{rx} + \frac{t_{pkt}}{\lambda} \tag{3.1}$$

Note that d_{ij} is minimized for $\lambda = 1$. Therefore, *it is more efficient to use preambles of same length as the wakeup period t_{rx} than to use shorter preambles and retransmit until a preamble hit.* This observation is illustrated in Figure 3.2. In one case, the sender uses $\lambda = 1$ and hits the receiver after one transmission, for a cost of

---

[2] This is true under our assumption of reliable (or filtered) links in conjunction with the asynchronous duty cycling. Of course with unreliable links, a preamble hit is a necessary but not sufficient condition for a successful packet reception.

$d_{ij} = t_{rx} + t_{pkt}$. In the other case, the sender uses shorter preambles with $\lambda = 0.5$, but must make three transmissions until the first preamble hit, for a total cost of $d_{ij} = \frac{3}{2}t_{rx} + 3t_{pkt}$.

Of course, we see in (3.1) that if t_{pkt} is very short relative to t_{rx}, as is the case for very low duty cycles (Table 2.2), then the impact of λ on d_{ij} is reduced: the increase in transmission cost with using $\lambda < 1$ is proportional to the packet length t_{pkt}. In the extreme, if $t_{pkt} = 0$, we would theoretically have no overhead in using preambles shorter than t_{rx} and retransmissions. Still, the key point remains that there is no theoretical advantage to reducing preambles in the unicast case, and in practice there is even a disadvantage to doing so, because switching the radio from transmit to receive mode and vice-versa, has a cost.

The reason for a detour to show that a naive strategy such as this one is not worthwhile may appear surprising to the reader at this point. Its motivation shall become apparent in the next chapter, where we will see that the situation reverses in the case of anycast forwarding, where a significant gain can be had from reducing preamble lengths.

On energy cost vs node lifetime

As we have seen in Table 2.1, the overall energy requirement of a wireless link layer is determined not only by the time spent transmitting, but also by the time spent listening to the channel and receiving packets. Note that with the energy cost metric defined above for asynchronous duty cycling, finding a shortest path corresponds to finding the path for which total *transmission* costs will be minimized. This path cost does not reflect the cost of idle listening, which is natural (in the asynchronous duty cycling model) since idle listening is a periodic and continuous process, that takes place independently of routing.

With this modelling , we are choosing to fix one parameter (duty cycle, which is proportional to the energy used to *listen* to the channel), and then minimize transmission cost. We could equivalently have chosen to fix transmission cost, and minimize duty cycle. This point raises a fundamental design question: is the "optimal"[3] design of a link layer and/or routing protocol separable in this sense? We do not argue that this is the general case. In fact there are certainly examples where it is *not* the case.

However, finding the optimal *joint* design requires making precise assumptions on traffic patterns (both temporal and spatial), network topology, and node distribution, in order to find paths and duty cycling disciplines that balance load evenly among

---

[3] Where optimal may for example be defined as maximizing expected node lifetime for a given traffic and topology model.

nodes. However history has shown that ex-cathedra assumptions on traffic patterns often turn out to be wrong, and the same can be expected to hold for network topologies. Finally, any design which is finely tailored to a specific operating regime is likely to suffer from rapidly degrading performance if the effective regime turns out to be different, whereas a more general design, which is not optimal for any given regime, is likely to be more flexible and have "good" performance over a wider range of conditions.

3.5 Single-Path Routing and Bellman-Ford

While not strictly part of a network model, the Bellman-Ford algorithm for computing shortest paths is an essential foundation for many distributed routing protocols. For this reason, and because we shall later use an algorithm derived from Bellman-Ford to construct anypath routes, we review this algorithm here.

The function of a routing algorithm is to establish a route from a source to a destination given the graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ that the source and destination inhabit. A first requirement of the algorithm is *correctness*: routes should terminate at the destination and be loop-free. Of course, many loop-free routes between two nodes are possible, and simply assuring that a packet eventually arrives at the destination is not enough. Accordingly, single-path routing algorithms usually seek to find the *shortest path* between each node and the destination. As a consequence, when we refer to single-path routing in this dissertation we shall implicitly mean routing over *shortest* path routes. Of course, which path is the shortest depends on the link and path metrics employed; as examples we may use the ones outlined in the previous section[4].

Concretely, a routing algorithm must compute, at every node, the *next-hop* neighbor to whom packets for a given destination should be forwarded. Selecting the best next-hop amongst all neighbors requires knowledge of the distance between each neighbor and the destination. Conversely, the problem of computing the distance from a node to the destination is equivalent to finding the shortest path itself, and summing the distances of the constituent links along the path.

This duality between computing the shortest-path distances and the shortest-path routes themselves is reflected in the Bellman-Ford algorithm, which jointly updates at each iteration both the distance to the destination and the next hop. We denote

---

[4] In large, heterogeneous networks such as the Internet, policies and agreements between network providers play an important role in route selection and path lengths are not the only criterion; we do not consider these external factors in the context of wireless networks operating under a single entity.

D_i^h to be the distance from i to the destination at the h-th iteration and N_i^h to be the next hop, with initial conditions $D_i^1 = \infty$. Assume without loss of generality that the destination is node 1 and take by convention $D_1^h = 0$ for all h. Then one iteration of the Bellman-Ford algorithm consists of finding at each node i the next-hop among i's neighbors that minimizes the path length from i to the destination, and computing the resulting path length from i.

$$\begin{aligned} D_i^{h+1} &= \min_{j \in N(i)} [d_{ij} + D_j^h], \\ N_i^{h+1} &= \arg \min_{j \in N(i)} [d_{ij} + D_j^h], \qquad \text{for all } i \neq 1. \end{aligned} \tag{3.2}$$

This algorithm can be shown to terminate in at most $|\mathcal{N}|$ iterations and compute the shortest paths from every node to the destination. When it converges, we have $D_i^h = D_i^{h-1}$ for all i, and the solution satisfies the Bellman equation:

$$D_i = \min_{j \in N(i)} [d_{ij} + D_j], \qquad \text{for all } i \neq 1. \tag{3.3}$$

This breakdown of a route cost between the next-hop cost d_{ij} and the distance D_j from the next-hop to the destination is illustrated in Figure 3.3.

The iteration (3.2) is inherently amenable to a distributed implementation, since the update step at each node i takes as input only state that is local to i's neighborhood. Therefore, if nodes know the distances d_{ij} to their neighbors, and if nodes periodically make local broadcasts containing their estimated D_j^h, each node has sufficient knowledge to estimate their D_j^{h+1}. Under a few technical assumptions stating essentially that the time interval between successive computations of (3.2) at each node is finite, and that old distance information is eventually purged from the system, this distributed implementation can be proven to converge to the shortest paths in finite time. We refer to the excellent presentation of Bertsekas and Gallager in [7] for an in-depth coverage of the Bellman-Ford algorithm.

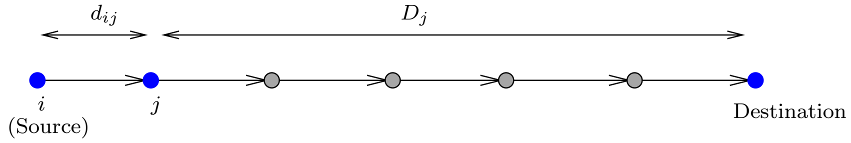


Figure 3.3: Breakdown of single-path route cost. The cost of the route from the source i to the destination can be broken down into the cost d_{ij} to reach the next hop j, and the cost D_j to then reach the destination from node j. The Bellman-Ford algorithm selects at each node the next-hop k minimizing the sum $d_{ij} + D_j$.

3.6 Summary

Link model	Link Metric d_{ij}	Metric Type	Metric Interpretation
Always-on unreliable links	$-\log p_{ij}$	Delivery probability	Probability that j receives packet after one transmission.
Always-on unreliable links with ARQ	$\frac{1}{p_{ij}}$	Transmission count	Expected number of transmissions until j receives packet.
Asynchronous duty cycled links with ARQ	$t_{rx} + t_{pkt}$	Transmission energy	Energy cost to send a packet to j.
Clustered duty cycling (synchronous)	$t_{rx}/2$	Latency	Expected delay to send a packet to j.
Individual duty cycling (synchronous)	$t_{rx}/2$	Latency	Expected delay to send a packet to j.

Table 3.2: Summary of link models, associated unicast link costs, and their interpretation.

This chapter has defined the key link-layer models and costs that are used in the remainder of this dissertation. Two of these models are for *always-on* wireless links. With always-on links, we use either the delivery probability (E2E) as metric, or the expected transmission count (ETX) metric, for link-layers with retransmissions. The other three models are duty-cycled links. The first is asynchronous duty-cycling, where node schedules are non-coordinated; the next two are models of synchronous duty cycling, where nodes are informed of their neighbors' schedules.

Table 3.2 summarizes these link models, their associated unicast costs, and their interpretation.

Chapter 4

Anycast Forwarding

Notation and Key Concepts

$C(i)$	Candidate relay set (CRS) Neighbors of i which may be used as relay nodes for packets forwarded by i (toward a given destination). Generalizes the notion of next-hop in single-path routing.
$A(i)$	Available relay set (ARS) Nodes in $C(i)$ that are *available* to forward a given packet. $A(i) \subseteq C(i)$. Anycast forwarding is *sender-driven* if the ARS is known to the sender *before* the packet transmission. Anycast forwarding is *receiver-driven* if the ARS is known to the sender only *after* the packet transmission.
ERS	Effective Relay Selection Policy to select effective relay for a given packet transmission where $\|A(i)\| > 1$.
d_{iJ}	Anycast link cost (ALC) Cost to send a packet from node i to any node in the set J. Generalizes the notion of unicast link cost.
R_{iJ}	Remaining path cost (RPC) Expected cost to reach the destination from a node in the CRS J. Generalizes the single-path notion of distance from the next-hop to the destination.

In Chapter 1, we illustrated with some simple examples how anypath routing could take advantage of having multiple candidate relays at each hop in order to increase throughput or decrease packet loss. Our point of view was a global one, and we considered the entire path between a source and destination. Using contrived topologies for clarity of exposition, we were able to directly find the best anypath route and compute its cost. The aim of this dissertation is to develop a general framework of theory and algorithms applying the intuition of these examples to general topologies. The first step in this development requires us to take a microscopic, local viewpoint, and examine in detail what happens at the level of a single hop.

Specifically, this chapter is organized as follows. We first introduce the notion of *candidate relay set* (CRS), which generalizes the next-hop of single-path routing to be a set of possible next-hop nodes. With multiple candidate relays, we must distinguish between a sender that is able to select the intended relay before transmission, and a sender that is not. We call these two situations *sender-driven* versus *receiver-driven*[1] anycast forwarding. In the case of receiver driven forwarding, multiple nodes may receive a packet, and if we want to avoid having duplicate relays, we need a policy to select the node that is to be the relay among those that received it. A natural policy that will be assumed in this chapter is to choose the relay with the lowest cost to the destination; we shall consider other policies in Chapter 6.

Having introduced these notions, we then define the cost metrics that will drive anypath routing. The first is the *anycast link cost* (ALC), which generalizes the unicast link cost that is used by single-path routing. The ALC is the cost to send a packet to *any one* node in a candidate relay set; it should be lower than the cost to send a packet to one *specific* node, or else there will be no advantage to anypath routing over single-path routing. After the anycast link cost we introduce the notion of *remaining path cost*, which intuitively corresponds to the expected path cost to the destination when using a given candidate relay set.

Finally in the third part of this chapter we develop the anycast link cost and remaining path cost in the context of the wireless network models from Chapter 3, first for always-on networks, and then for duty-cycled networks.

Summary of contributions

This chapter provides a definition of the mechanisms and costs of anycast forwarding, the foundation upon which builds anypath routing. Specifically, these contributions include:

- Defining the cost of anycast forwarding and the remaining path cost,
- Demonstrating, with new link layer mechanism, how anycast forwarding can increase energy efficiency or decrease latency with low-power duty-cycled links, and
- Distinguishing between *sender-driven* and *receiver-driven* anycast forwarding, and the role of Effective Relay Selection (ERS), and showing that anycast for-

---

[1] Our use of the term receiver-driven is quite different from its earlier use in separate networking topics such as multicast rate adaptation [74].

warding is useful not only with broadcast-based links, but also with unicast links.

4.1 Link-Layer Anycast

4.1.1 Candidate Relay Set

The core feature underlying anypath routing is the use of *anycast forwarding.* Anycast forwarding is a link-layer mechanism and therefore spans a single hop. Its semantics are similar to those of anycast routing [104], but operating over a single hop: a packet is to be sent to *any* one node out of a set of neighbors. This is in sharp contrast with unicast packet forwarding, where a packet is sent to a single neighbor, and with broadcast forwarding, where a packet is sent to *all* neighbors. The use of anycast forwarding is what makes anypath routing fundamentally different from single-path or multi-path routing.

When a sender i transmits a packet using anycast forwarding, it might be acceptable for any node out of its *entire* neighborhood $N(i)$ to receive the packet. In the context of anypath forwarding, the nodes that a packet is anycast forwarded to are those nodes that may in turn relay the packet further. We usually want to allow only certain specific nodes of $N(i)$ to relay the packet further, and so the set of anycast receivers is usually smaller than $N(i)$. We call this set the *candidate relay set* (CRS), and Figure 4.1 illustrates some possible choices of this CRS.

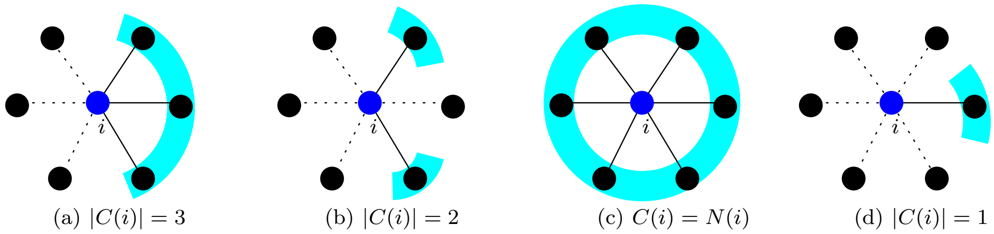
Definition 4.1. *The* Candidate Relay Set *(CRS) $C(i)$ of a node i is a subset of i's neighbors containing all the nodes which may be used as relay nodes for packets forwarded by i toward a given destination*[2]*. An anypath routing protocol assigns one candidate relay set per destination to each node.*

Just as a single-path routing protocol computes the next hop relay node from each node toward the destination, the function of an anypath routing protocol shall be to compute, at each node, and for a given destination, the candidate relay set. Candidate relay selection thus naturally generalizes the next-hop decision of a single-path algorithm: instead of choosing a *single* next hop, we choose a *set* of potential next hops.

How do we choose which neighbors may be used as relays? Intuitively, identifying the candidate relays in an arbitrary network requires finding all neighbors that are "better placed" than the transmitter to reach a given destination, whilst ensuring that

---

[2] Recall that for simplicity of notation we shall consider a *single* destination, and thus the notation $C(i)$ does not make reference to the destination.



(a) $|C(i)| = 3$ (b) $|C(i)| = 2$ (c) $C(i) = N(i)$ (d) $|C(i)| = 1$

Figure 4.1: Different choices of candidate relay sets. The sender node i is in the center and is surrounded by 6 neighbors. The role of an anypath routing protocol is to assign a candidate relay set to each node in the network.

no cycles are present in the resulting graph. Of course deciding if a node is "better placed" requires having some notion of distance from each node to the destination, and this distance is computed by the routing protocol itself. Just as the Bellman-Ford algorithm in the single-path case jointly computes the distance and the next-hop node from each source to the destination, our anypath routing algorithm jointly computes the distance and the *candidate relay set* from each source to the destination, taking the same link costs to each neighbor into consideration. Unsurprisingly, the notion of distance used by anypath routing shall turn out to be quite different than the path distances used in single-path routing; this topic is at the heart of the following chapter.

Routing, forwarding, and the Available Relay Set

Before proceeding further it is important to distinguish between *routing* and *forwarding*. Routing is the process of computing which path(s) a packet should go through to reach a particular destination. In practical terms, routing populates the routing table at each node with information about the next hop relay(s) for a given destination. It is therefore a control function which happens independently of data traffic[3].

The term *forwarding* refers to the process of moving transit packets in and out of a wireless node. The forwarding process includes looking through the routing table, making the forwarding decision, placing the next-hop address in a packet header, and sending the packet out of an interface.

It shall be necessary in the following paragraphs to distinguish between the candidate relay set $C(i)$, and a subset $A(i) \subseteq C(i)$ of the CRS. This subset is called the *available relay set* (ARS), and contains the nodes in the candidate relays that are able

---

[3] Some wireless routing protocols establish routes *on-demand* [46, 82] by flooding control packets when needed; these are used mostly in mobile scenarios and are not considered in this dissertation.

to forward *a given packet.*

Definition 4.2. *The* Available Relay Set *(ARS)* $A(i)$ *of a node* i *is a subset of* i*'s candidate relay set* $C(i)$ *containing all nodes that are* available *to forward a given packet. The ARS may be different for each successive packet.*

Unlike the CRS, the available relay set is not a fixed set chosen by the routing protocol, but is a random subset that potentially changes for each forwarded packet. In the simplest case, the ARS is static and is always equal to the CRS. At the other extreme, the ARS may change on a per-packet basis (e.g., if links are fluctuating rapidly and all nodes in $C(i)$ are rarely reachable at the same time).

4.1.2 Sender-Driven versus Receiver-Driven Forwarding

Anycast forwarding can be either *sender-driven*, meaning that the next-hop node is determined before the transmission (as in most routing protocols), or it can be *receiver-driven*, meaning that the next-hop is decided after transmission. In the receiver-driven case, the sender transmits a packet that may be received by any of the nodes in the CRS; the decision of which node(s) becomes the effective relay is made after the transmission has taken place.

Definition 4.3 (sender- and receiver-driven anycast forwarding)**.** *Anycast forwarding is* sender-driven *if the ARS is known to the sender* before *the packet transmission. Anycast forwarding is* receiver-driven *if the ARS is known to the sender only* after *the packet transmission.*

We illustrate this definition with a few examples. Some examples of sender-driven forwarding are:

- A wired network where a router can select an outgoing port for transmission.
- A peer-to-peer network with random node disconnections; if nodes keep track of the status of their neighbors, then node i knows which candidate relays in $C(i)$ are available, and it can choose one that is connected to send its packet to.
- A wireless network with side information: pairwise channels are either in outage or are working reliably, and the sender knows *before* sending in which of these two states each channel is.

Receiver-driven forwarding happens essentially when there is uncertainty at the sender about the outcome of a transmission, because the sender cannot forecast in advance channel conditions, or because of unpredictable neighbor outages. In this case the relay must be chosen *after* the transmission of a packet. Some examples are:

- A peer-to-peer network with random node disconnections; if nodes *do not* keep track of the status of their neighbors, then node i does not know before sending which candidate relays in $C(i)$ are available, and so it cannot choose the relay before sending.

- A wireless network with an unreliable broadcast channel to all neighbors, if the reception at each neighbor are independent, the sender does not know *before* transmitting which neighbor will receive the packet, and thus cannot select the relay from $C(i)$ before sending.

Discussion

The difference between sender- and receiver-driven forwarding is a subtle and multifaceted one. We discuss this difference using as a guiding thread some alternate terms that could have been adopted instead to distinguish both types of forwarding.

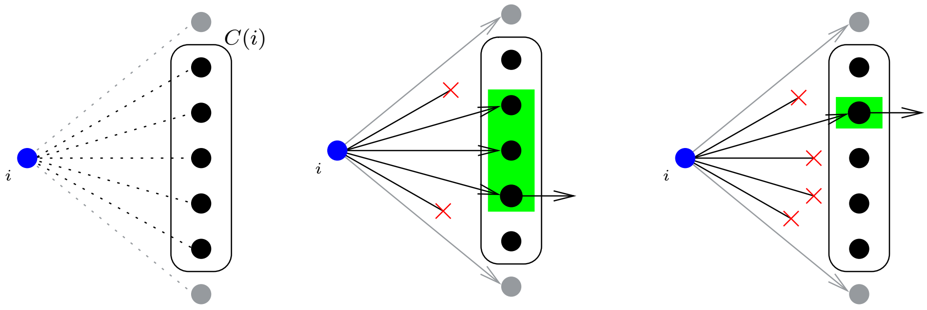
Morris et al [9] use the term *opportunistic* routing to refer to an early form of anypath routing. Their use of this term is intended to convey the fact that the forwarding mechanism is able to greedily take advantage of any receiver, *after* a packet has been transmitted. Their work contains no notion corresponding to our sender-driven forwarding. In fact, the authors explicitly make the case that selecting the next-hop after the packet is forwarded is a fundamental requirement of this class of routing schemes, which this dissertation shows not to be the case.

One might also surmise that sender- and receiver-driven anycast forwarding are tied respectively to unicast and broadcast communication media: with unicast the sender can choose its destination, whereas with broadcast the sender transmits and any node in $C(i)$ may receive the packet. This is also incorrect, as the preceding examples show.

Finally, we could also use the terms *deterministic* versus *random* forwarding, because in one case the sender knows $A(i)$ and in the other $A(i)$ is unknown to i until the dice (or more accurately, the packet) has been thrown. However, this would obscure the fact that the set $A(i)$ for a given packet may be entirely deterministic; and the apparent randomness may simply be an artifact of lack of information at the sender i.

4.1.3 Effective Relay Selection

The examples given previously to illustrate sender- and receiver-driven forwarding showed that in either case, the available relay set $A(i)$ may contain multiple nodes from the candidate relay set $C(i)$. In the sender-driven case, we must choose the effective relay from $A(i)$ *before* sending, and in the receiver-driven case, we must



(a) (b) $|A(i) = 3|$ (c) $|A(i) = 1|$
(ERS not needed)

Figure 4.2: Illustration of available relay sets (ARS) and effective relay selection (ERS). (a) Sender node i with candidate relay set $C(i)$ containing 5 nodes. (b) Node i sends a packet that is received by three nodes in $C(i)$. These three nodes form the available relay set $A(i)$, and the ERS policy selects the lower one to be the next relay. (c) If only one node in $C(i)$ receives the packet, we have $|A(i)| = 1$ and the choice of relay is implicit.

choose the relay *after* sending. Of course in either case it may happen that there is only one node to choose from ($|A(i)| = 1$), but in general it is possible that we must choose among multiple nodes. This choice is made using a policy that we call *effective relay selection*. Figure 4.2 illustrates both the concepts of available relay set and effective relay selection.

Definition 4.4. *The* Effective Relay Selection *(ERS) policy chooses, for each packet transmission where $|A(i)| > 1$, the node(s) in $A(i)$ that should effectively relay the packet.*

Note that unlike the candidate relay set, the notion of ERS policy has no analogue in single-path routing: with a single possible next-hop node, there is no such decision to be made (because $|C(i)| = 1$, and $0 \leq |A(i)| \leq 1$). There are even certain specific cases of anypath routing where the ERS policy is unnecessary. As an example in the sender-driven case, consider a wireless network where i has n orthogonal unicast channels to each node in $C(i)$, for example with time-division multiple access (TDMA). If the TDMA schedules are non-overlapping, then we never have $|A(i)| > 1$, and an ERS decision is never required.

Before discussing possible ERS policies, a few remarks are in order:

- It may appear at first glance that the ERS policy is only relevant to *forwarding*, since it is carried out on a per-packet basis as part of anycast forwarding. However, we shall see in Chapter 5 that it is also directly relevant to *rout-*

ing, because a routing decision that selects "good" candidate relay sets requires knowing which ERS policy shall be used.

- From a conceptual or algorithmic standpoint, there is no difference between ERS for sender or receiver-driven forwarding: in either case, we must choose the effective relay(s) out of $A(i)$.
- From a protocol or implementation standpoint, there is a significant difference between ERS for sender or receiver-driven forwarding. While sender-driven forwarding simply requires transmitting a packet to the relay(s) in $A(i)$ chosen by the ERS, carrying out the ERS decision with receiver-driven forwarding requires a protocol mechanism to reach agreement (between the sender and all nodes who received the packet) on which node(s) will forward it further. Carrying out this arbitration may come at a non-negligible cost.

We shall consider three ERS policies in this dissertation. The first policy corresponds to the intuition that one would like to select the node in $A(i)$ that is "best placed" to become the effective relay. We call this policy *ERS-best*, and will assume its use exclusively throughout this chapter. Again, like for deciding the candidate relay set, this requires a notion of distance to the destination.

Definition 4.5 (ERS-best)**.** *The* ERS-best *policy selects as effective relay the node j with minimal distance to the destination out of all available relays:*

$$j = \arg\min_{k \in A(i)} D_k$$

The second and third ERS policies shall be examined in the following chapter, and are chiefly of interest in conjunction with receiver-driven forwarding, because they lead to simpler implementations than ERS-best. The second policy is simply to choose one node at random in $A(i)$ to become the effective relay. We call this policy *ERS-any*. It may appear surprising at this point to bother defining ERS-any, since ERS-best by definition selects better paths. However, the cost of implementing ERS-any may be significantly lower, and so even if it sometimes selects suboptimal relays, the overall cost cannot be a priori ruled to be higher.

Other policies are possible, in particular ERS policies that allow *multiple* receiving nodes to relay a packet. The third ERS policy that we shall consider is part of this family and is called *ERS-all*. With ERS-all, *every* node in $A(i)$ is chosen to be a relay. This potentially results in multiple relays of a given packet, and may lead to an uncontrolled explosion of transmissions, in particular if an anypath route has a large fan-out. It may thus appear from the outset that there is no reason to even consider ERS-all. However, implementing ERS-all in a protocol mechanism has even lower cost than ERS-any, and so we cannot a priori discard it from our evaluation.

4.2 Anycast Link Cost

In order to design and analyst anypath routing algorithms that compute the best candidate relay set at each node, we must define the notions of cost that drive such algorithms. In this section, we introduce the notion of anycast link cost d_{iJ} to send a packet from i to any node in the set J. It shall be implicit throughout this dissertation that the set J is a subset of node i's neighbors ($J \subseteq N(i)$) whenever we use the notation d_{iJ}.

Definition 4.6 (Anycast link cost)**.** *The* anycast link cost *(ALC) from node i to neighbors J, denoted d_{iJ}, is the cost to send a packet from i to* any *node in the set J, where $J \subseteq N(i)$ is a subset of i's neighbors.*

The anycast link cost is a generalization of the link cost defined in Section 3.4. On a note of terminology, the term "link cost" shall continue to refer to the unicast link cost of Section 3.4; when the context is ambiguous we explicitly call it *unicast* link cost.

Similarly to unicast link costs, the anycast link cost can be assigned in a number of ways, depending on the cost criterion and problem we are interested in. Before we examine in more detail the anycast link cost and its definition under different link models, let us first state two simple properties that it should have. The first is that the anycast link cost to set of size 1 should be equal to the unicast link cost to the single node in the set.

Property 4.1. *If $J = \{j\}$, then $d_{iJ} = d_{ij}$.*

The second property is that the anycast link cost to a set containing the sending node itself is 0; it is a generalization of the unicast link cost property requiring that the distance from a node i to itself is $d_{ii} = 0$.

Property 4.2. *If $i \in J$, then $d_{iJ} = 0$.*

These two properties will be defined to hold by convention and we will not repeat them in the definitions of all the anycast link distances given in this chapter.

Beyond these two trivial requirements, the anycast link cost can in theory be any function of the set J. The key is to define a function that captures the cost in a way that is in agreement with the cost model in the unicast scenario. Of course, for anypath routing to be worthwhile, we are interested in anycast link costs which *decrease when the candidate set is expanded*. In other words, we would like to have:

$$d_{i(J \cup j)} < d_{iJ}. \tag{4.1}$$

We say that an anycast link cost is *always-decreasing* if it satisfies the above equation:

Property 4.3. *An anycast link cost is* **always-decreasing** *if it satisfies* (4.1) *for all* $i \in \mathcal{N}$*,* $J \subset N(i)$*, and* $j \in N(i) \setminus J$

Note that the always-decreasing property is a rather strong one, since it contains a strict inequality. One might conceivably come across anycast link distances which decrease for the addition of *some* but not *any* nodes to the set J. For example, an anycast link cost may reach a minimum when J contains 5 nodes and not decrease thereafter as $|J|$ grows beyond 5. Or, independently of the size $|J|$, an anycast link cost may decrease only with the addition of certain nodes. For example, if the loss patterns on the links (i, j) and (i, k) are identical, then there would be no reduction of loss probability by adding *both* j and k to J, whereas if the loss patterns are independent, there would be a reduction. A weaker property, which is implied by always-decreasing, is therefore *sometimes-decreasing*:

Property 4.4. *An anycast link cost is* **sometimes-decreasing** *if it satisfies*

$$d_{i(J \cup j)} \leq d_{iJ},$$

for all $i \in \mathcal{N}$*,* $J \subset N(i)$*, and* $j \in N(i) \setminus J$*.*

Observe that if an anycast link cost is neither always-decreasing nor sometimes-decreasing, and in particular if d_{iJ} is minimized when J is a singleton, then there is no gain to be found with anypath routing. We call this a *unicast-equivalent* link cost metric:

Property 4.5. *An anycast link cost is* **unicast-equivalent** *if it satisfies*

$$d_{iJ} = \min_{j \in J} d_{ij}$$

for all $i \in \mathcal{N}$*,* $J \subset N(i)$*, and* $j \in N(i) \setminus J$*.*

A trivial example of a unicast-equivalent cost metric is expected transmission count (ETX, defined in Chapter 3, page 46) in conjunction with reliable links. Here, the distance (in hops) to reach *any* neighbor in J is 1, just as the distance to one specific neighbor is also 1. This example should not lead one to think that all wired network models have costs that are are unicast-equivalent. For example, in a peer-to-peer network where peers are frequently disconnected from the network, and if we consider the ALC to a set of peers J to be the probability that at least one node in J is up, then the ALC is at least sometimes-decreasing, and possibly always-decreasing (depending on the specific assumptions of this model).

Properties 4.3, 4.4, and 4.5 are qualitative ones. While they are easy to show for different anycast link costs and give a high-level characterization of an ALC's behavior, they do not tell us how quickly or how far d_{iJ} will decrease with J. One quantitative manner to characterize an anycast link cost d_{iJ} is to compare it with the lowest unicast link cost between i and any node in J:

Definition 4.7 (Anycast forwarding gain)**.** *The* anycast forwarding gain $G(d_{iJ})$ *is the ratio*

$$G(d_{iJ}) = \frac{d_{iJ}}{\min_{j \in J} d_{ij}}$$

of the anycast forwarding cost from i to J to the lowest unicast cost from i to any node in J.

This gain is therefore 1 for a unicast-equivalent ALC. In the general case, the anycast link cost depends on the specific choices of i and J. However, in many cases of interest, it depends only on the size of $|J|$. In these cases, we can characterize the scaling behavior of the anycast forwarding gain, i.e., how fast it decreases with the size of the candidate relay set.

4.3 Remaining Path Cost

In the first part of this chapter we introduced the concept of anycast link, and defined its cost (ALC) under different network and cost models. We saw that in network models of interest, anycast links have lower costs than the corresponding unicast links.

Consider now a node i whose candidate relay set $C(i)$ is given to us. Assume also that we know the costs (or distances) D_j to reach the destination from each node $j \in C(i)$. How do we now compute the cost (distance) D_i from i to the destination? Clearly it should include the anycast link cost from i to $C(i)$. And it is also clear that it should include in some way, the costs from nodes in $C(i)$ to the destination, since each packet from i will end up being forwarded by one of the nodes in $C(i)$. Intuitively, D_i should be the sum of the anycast link cost from i to any node in $C(i)$ and the cost to reach the destination from one of the nodes in $C(i)$. Figure 4.3 illustrates the breakdown between anycast link cost and remaining path cost. We can

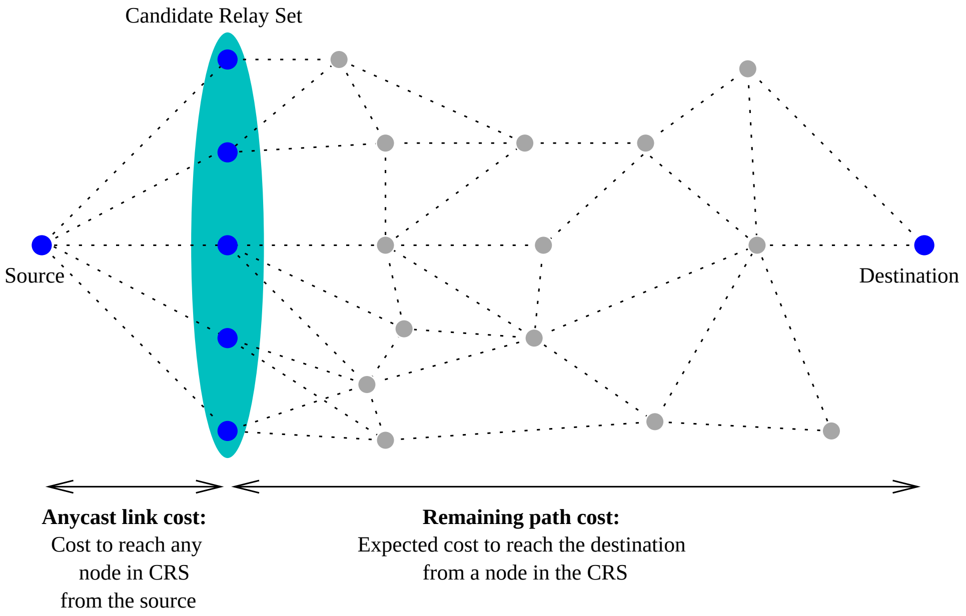


Figure 4.3: Breakdown of anypath route cost. A path cost can be broken down into two parts: the cost to reach any node in the candidate relay set, and the cost to reach the destination from the candidate relay set. These two costs generalize respectively the costs d_{ij} and D_J of single-path routing, shown in Figure 3.3 (p. 52).

contrast it with the equivalent concepts in single-path routing (illustrated in Figure 3.3, p. 52) as follows:

- In single-path routing the distance from a node i to the destination, given that node i is using node j as a next hop relay, is the sum $D_i = d_{ij} + D_j$ of the unicast link distance from i to j with the unicast path distance from j to the destination.
- In anypath routing, the distance from a node i to the destination, given that node i is using $C(i)$ as a candidate relay set, is the sum of the anycast link distance from i to $C(i)$ and the remaining path cost from $C(i)$ to the destination: $D_i = d_{iC(i)} + R_{iC(i)}$, where we have used the term $R_{iC(i)}$ to represent the (as yet undefined) remaining path cost.

Based on the intuition outlined above, we can define the remaining path cost as follows:

Definition 4.8 (Remaining path cost)**.** *The* remaining path cost R_{iJ} *is the expected cost to send a packet to the destination after it has been forwarded from i to the* *candidate relay set* J*.*

Note that the remaining path cost R_{iJ} does *not* include the anycast link cost from i to J. It may therefore appear that it is a function of the candidate relay set alone (e.g., a weighted average over the costs of nodes in J), and that the subscript i in R_{iJ} is superfluous. In fact, in many cases the remaining path cost *does* depend on i as well as J, since the delivery probabilities from i to nodes in J influence the probability that any given node in J becomes the effective relay. Of course in the single path case, with $J = \{j\}$, the remaining path cost reduces simply to the cost D_j to send a packet from node j to the destination, and it is therefore independent of the sending node i.

Let us consider the outcome of the anycast forwarding phase after node i has forwarded a packet. The anycast forwarding phase consists of a single packet transmission if we are not using ARQ, or, if we are using ARQ, a sequence of transmissions that stops as soon as one node in the anycast set receives the packet. Let us now define the random vector $\mathbf{A} \in \{0,1\}^{|C(i)|}$ representing the outcome of the anycast forwarding phase, whose k-th component $\mathbf{A}_k$ is an indicator random variable equal to 1 if the k-th node in $C(i)$ received a packet forwarded by i and 0 otherwise. Equivalently, $\mathbf{A_j} = \mathbf{1}$ iff $j \in A(i)$ for this packet. Under this notation, the random variable $\mathbf{1}^T\mathbf{A}$ is greater than zero when the available relay set is non-empty (for a given packet). Note the following regarding the events $\{\mathbf{A_j}\}$:

- First, $\mathrm{P}(\{\mathbf{A_j} = \mathbf{1}\})$ is not necessarily equal to p_{ij} in the general case: we are looking at the receivers after a *complete* anycast forwarding phase has taken place, and not simply after node i transmits a single packet. Whether or not the two situations are the same depends on the link model we are considering, and specifically, on the use (or not) of ARQ. Without ARQ, the anycast forwarding phase consists exactly of one packet transmission from node i, and so $\mathrm{P}(\{\mathbf{A_j} = \mathbf{1}\}) = p_{ij}$. With ARQ, the anycast forwarding phase may consist of more than a single transmission from i, and so the two probabilities may be different.

- Second, it is possible that *no* node in $C(i)$ receives the packet after the anycast forwarding phase. In other words, it is not true in the general case that $\mathrm{P}\left(\mathbf{1}^T\mathbf{A} > 0\right) = 1$. Whether or not $\mathrm{P}\left(\mathbf{1}^T\mathbf{A} > 0\right) = 1$ again depends on the link model. Specifically, with unreliable links and no ARQ, $\mathrm{P}\left(\mathbf{1}^T\mathbf{A} > 0\right)$ is less than 1 (unless of course $p_{ij} = 1$ for some node j in $\in C(i)$). With ARQ, $\mathrm{P}\left(\mathbf{1}^T\mathbf{A} > 0\right) = 1$ (recalling that our model considers an idealized ARQ with which unreliable links become completely reliable).

We now define $f : \{0,1\}^{|C(i)|} \rightarrow \mathbb{R}$ as the function assigning to a realization of $\mathbf{A}$ the cost of the effective relay that is chosen. For example, in the particular case where $\mathbf{A}$ has a single non-zero component in the k-th position, then $f(\mathbf{A}) = D_k$.

In the general case however, the function f also depends on the effective relay policy (ERS-best or ERS-any). For example, if $\mathbf{A}$ has two non-zero components in the j-th and k-th positions, then $f(\mathbf{A}) = \min(D_j, D_k)$ under the ERS-best policy. Under the ERS-any policy, $f(\mathbf{A})$ would be a random variable taking either of the values D_j and D_k with equal probability 1/2. To define the function completely over its domain, we add that $f(\vec{0}) = 0$.

With these notations, we can now formally define the remaining path cost $R_{iC(i)}$ as the expected cost to send the packet from the next hop to the destination, conditional on at least one node in the candidate relay set receiving the packet:

$$R_{iC(i)} = \mathrm{E}\left[f(\mathbf{A})|\mathbf{1}^T\mathbf{A} > 0\right].$$

Noting $n = |C(i)|$, we can develop $R_{iC(i)}$ as follows:

$$\begin{aligned}
R_{iC(i)} &= \mathrm{E}\left[f(\mathbf{A})|\mathbf{1}^T\mathbf{A} > 0\right] \\
&= \sum_{\mathbf{A}\in\{0,1\}^n} f(\mathbf{A})\mathrm{P}\left(\mathbf{A}|\mathbf{1}^T\mathbf{A} > 0\right) \\
&\overset{(a)}{=} \sum_{\mathbf{A}\in\{0,1\}^n\setminus\vec{0}} f(\mathbf{A})\mathrm{P}\left(\mathbf{A}|\mathbf{1}^T\mathbf{A} > 0\right) \\
&= \sum_{\mathbf{A}\in\{0,1\}^n\setminus\vec{0}} f(\mathbf{A})\frac{\mathrm{P}\left(\mathbf{A},(\mathbf{1}^T\mathbf{A} > 0)\right)}{\mathrm{P}\left(\mathbf{1}^T\mathbf{A} > 0\right)} \\
&\overset{(b)}{=} \sum_{\mathbf{A}\in\{0,1\}^n\setminus\vec{0}} f(\mathbf{A})\frac{\mathrm{P}\left(\mathbf{A}\right)}{\mathrm{P}\left(\mathbf{1}^T\mathbf{A} > 0\right)} \\
&\overset{(c)}{=} \sum_{\mathbf{A}\in\{0,1\}^n} f(\mathbf{A})\frac{\mathrm{P}\left(\mathbf{A}\right)}{\mathrm{P}\left(\mathbf{1}^T\mathbf{A} > 0\right)} \\
&= \frac{1}{\mathrm{P}\left(\mathbf{1}^T\mathbf{A} > 0\right)}\mathrm{E}\left[f(\mathbf{A})\right],
\end{aligned} \tag{4.2}$$

where we used (a) the fact that $\mathrm{P}(\mathbf{A} = \vec{0}\,|\mathbf{1}^T\mathbf{A} > 0) = 0$, i.e. conditional on at least one node in $C(i)$ receiving the packet, the probability of no node in $C(i)$ receiving it is nil; (b) the fact that for $\mathbf{A} \neq \vec{0}$, the event "at least one node in $C(i)$ receives the packet" is included in the event $\mathbf{A}$; and (c) that $f(\vec{0}) = 0$ by definition. Note that in the above development, $\mathrm{P}\left(\mathbf{1}^T\mathbf{A} > 0\right)$ is the probability that at least one node in $C(i)$ receives a packet after the anycast forwarding phase (or equivalently, the probability that $A(i) \neq \emptyset$).

Remaining path cost with ERS-best

We define q_{ij} to be the probability that at the outcome of the forwarding phase, node j has received the packet from i. Equivalently, $q_{ij} = \mathrm{P}\left(\mathbf{A}_j\right)$. (As we shall see very soon, q_{ij} is equal to p_{ij} in some but not all models).

With the ERS-best policy, the effective relay is the node j with lowest cost D_j among all those in $A(i)$. Assume without loss of generality that the nodes in $A(i)$ are sorted by their distance to the destination, i.e., that $D_1 < D_2 < \ldots < D_j$. Then, the effective relay is node 1 if it received the packet (with probability q_{i1}), node 2 if it received the packet and node 1 did not, and more generally node k if it received the packet and nodes $1 \ldots k-1$ did not. The expected cost of the effective relay is then:

$$\begin{aligned}
\mathrm{E}\left[f(\mathbf{A})\right] = {} & q_{i1}D_1 + \\
& (1-q_{i1})q_{i2}D_2 + \\
& (1-q_{i1})(1-q_{i2})q_{i3}D_3 + \\
& \cdots \\
& (1-q_{i1})(1-q_{i2})\ldots(1-q_{in\text{-}1})q_{in}R_n \\
= {} & q_{i1}D_1 + \sum_{j=2}^{n}\left(\prod_{k=1}^{j-1}(1-q_{ik})\right)q_{ij}D_j.
\end{aligned} \tag{4.3}$$

A special case that we have seen in the previous section is when all nodes in $C(i)$ have the same probability of receiving a packet from i. Then, noting $q_{ij} = q$, the above expression simplifies to:

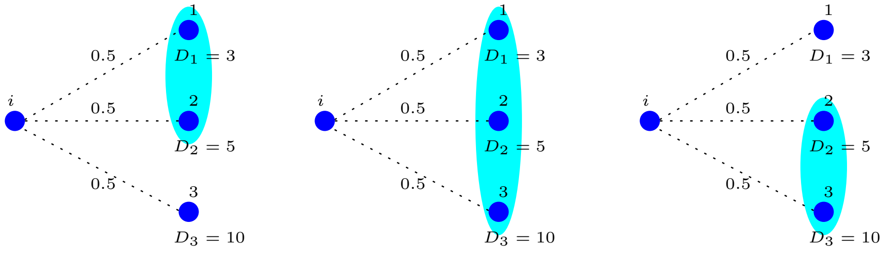
$$\mathrm{E}\left[f(\mathbf{A})\right] = q\sum_{j=1}^{n}(1-q)^{j-1}D_j. \tag{4.4}$$

Note that in this case, the remaining path cost depends only on the size of the CRS and the path costs D_j of nodes in the CRS. In particular, it *does not* depend on the forwarding node i.

Discussion and examples

In the single path case, it is immediate that the cost from a next hop j to the destination is simply D_j, and so it is not necessary to name the concept of remaining path cost as such. Nonetheless, the remaining path cost in anypath is once again a generalization of the single-path next-hop cost D_j, and when $J = \{j\}$, all of the remaining path costs computed in this section reduce to the cost D_j corresponding to the single-path case.

Just as in single-path, the remaining path cost depends on the next hop j and its distance D_j to the destination, in anypath it depends on the candidate relay set J and the distances $D_j, j \in J$ from each candidate relay to the destination. This is rather intuitive and expected. However, in the anypath case the remaining path cost is determined by more than J and the distances D_j: the remaining path cost depends also on the sender, that is, the node which forwards the packet to the candidate relay set.



(a) $C(i) = \{1, 2\}$ $R_{iC(i)} = 3.66$

(b) $C(i) = \{1, 2, 3\}$ $R_{iC(i)} = 4.57$

(c) $C(i) = \{2, 3\}$ $R_{iC(i)} = 6.66$

Figure 4.4: The remaining path cost depends on candidate relay set $C(i)$. Illustration of Example 4.1

Example 4.1 (Remaining path cost depends on CRS)**.** We illustrate the dependence of the remaining path cost on the CRS in Figure 4.4. The sender i is on the left, and we show three possible candidate relay sets highlighted in a shaded ellipse. Link delivery probabilities are annotated next to each link. Nodes use the policy ERS-best for relay selection.

Recall that the RPC is the expected cost from the effective relay to the destination. In Figure 4.4(a), we can directly compute the probability that either node is the effective relay. Conditional on at least one node in $\{1, 2\}$ receiving the packet, the probability that node 1 receives it is $\frac{2}{3}$, and the probability that node 2 receives it without 1 receiving it is $\frac{1}{3}$. With ERS-best, node 1 will always be preferred over node 2 if both nodes receive the packet. The remaining path cost is thus:

$$R_{iC(i)} = \frac{2}{3} D_1 + \frac{1}{3} D_2 = 3.66.$$

Note that this RPC is higher than if we had $C(i) = \{1\}$, in which case we would have $R_{iC(i)} = D_1 = 3$.

In Figure 4.4(b), the CRS consists of all three nodes $1, 2$, and 3. Node 2 is selected if it receives the packet and node 1 does not, and node 3 is selected if it alone receives the packet. We use equations (4.4) and (4.2) to compute the RPC:

$$R_{iC(i)} = \frac{p}{1-(1-p)^n} \sum_{j=1}^{n} (1-p)^{j-1} D_j = 4.57,$$

where $n = |C(i)| = 3$. The remaining path cost is *higher* than in the previous case, because we now allow the use of node 3, which has a higher distance to the destination. While node 3 will be used less frequently as the relay than 1 and 2 (because ERS-best only selects it when neither nodes 1 nor 2 receive the packet), its presence in the CRS still increases the remaining path cost to $R_{iC(i)} = 4.57$.

One should not immediately infer that the CRS of Figure 4.4(a) is a better choice just because it has a lower remaining path cost. Indeed, the choice of CRS should also take into account the anycast link cost, and the anycast link cost to the larger set is lower[4]. We explain in the next chapter how this tradeoff is at the heart of the anypath routing algorithm.

Finally, it should not come as a surprise that the CRS of Figure 4.4(c) has the highest remaining path cost of all three examples, because it takes as candidate relays the two nodes that are furthest from the destination. Here, the RPC is computed similarly to case (a) above:

$$R_{iC(i)} = \frac{2}{3} D_2 + \frac{1}{3} D_3 = 6.66.$$

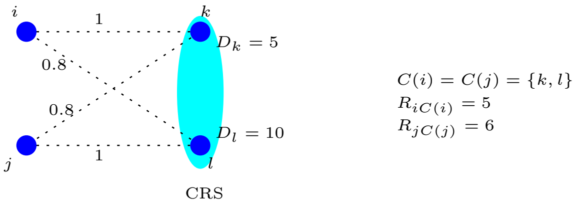


Figure 4.5: The remaining path cost depends on sender. Illustration of Example 4.2.

Example 4.2 (Remaining path cost depends on sender)**.** The dependence of the RPC on the sender is illustrated in Figure 4.5. Two forwarding nodes i and j each have the same candidate relay set $C = C(i) = C(j) = \{k, l\}$. Under ERS-best, node k is chosen as relay every time it receives a packet (because $D_k < D_l$), and node l is only chosen if it receives a packet and k does not. What are the remaining path costs R_{iC} and R_{jC} to the destination? Consider first sender i. Node k receives every

---

[4] At least for anycast link costs that are always-decreasing.

packet from i, and so is the effective relay for every packet (including for packets that node l had received as well). The remaining path cost R_{iC} is thus 5. The situation is different for sender j. Node k only receives packets from j with probability 0.8, whereas node l receives every packet, thus $R_{jC} = 0.8 * 5 + 0.2 * 10 = 6$.

4.4 Anycast Forwarding with Always-On Links

Having defined the cost of using an anycast link and the associated remaining path, we now show how these general definitions apply in wireless networks. We start with always-on networks.

4.4.1 Anycast Link Costs

We first examine the anycast link costs for always-on networks using unreliable links. As motivated in Section 3.4, we consider delivery and transmission cost metrics, but not energy metrics, since energy is usually not a central issue in an always-on network.

Let us first define the anycast link delivery probability, which generalizes the unicast link probabilities p_{ij} to the anycast case. The probability that a unicast packet from i is received by j becomes the probability that an anycast packet from i is received by *at least one* node in the set of nodes J. We call this the anycast link delivery probability to a set J, and its expression is:

$$p_{iJ} = (1 - \prod_{j \in J}(1 - p_{ij})).$$

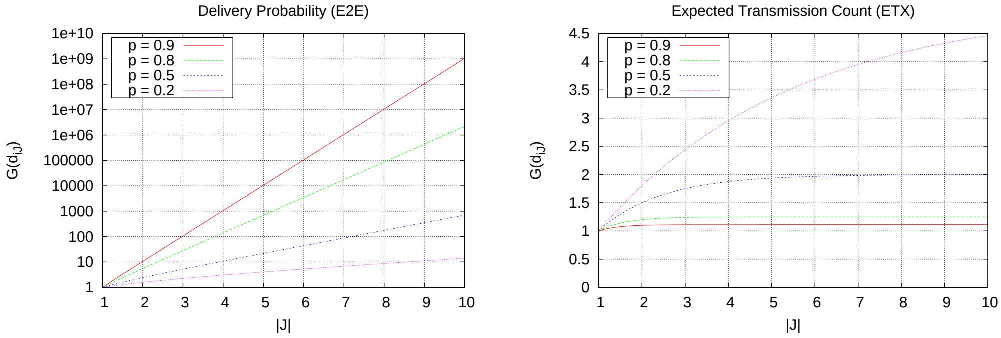
Note that the link delivery probability increases for every node that is added to the set J.

Delivery probability metric (E2E)

We first consider unreliable links without retransmissions. The anycast E2E metric generalizes the unicast case to the negative logarithm of the anycast link delivery probability:

Definition 4.9 (E2E anycast link cost)**.** *The delivery probability (E2E) anycast link cost for the unreliable link model is:*

$$d_{iJ} = -\log p_{iJ} \tag{4.5}$$



(a) (b)

Figure 4.6: Anycast forwarding gain for increasing candidate relay set size J. All links having same delivery probability p. (a) delivery probability cost metric (d_{iJ} as in Definition 4.9). (b) expected transmission count (ETX) cost metric (d_{iJ} as in Definition 4.10).

Transmission-count metric (ETX)

We now consider unreliable links with retransmissions. With anycast forwarding, the ETX metric of Section 3.4 generalizes to become the expected number of transmissions until *at least one* node in J receives the packet:

Definition 4.10 (ETX anycast link cost)**.** *The expected transmission count (ETX) anycast link cost for the unreliable link model with ARQ is the expected number of transmissions until a node in J receives a packet:*

$$d_{iJ} = \frac{1}{p_{iJ}} \tag{4.6}$$

We can immediately note from definitions 4.9 and 4.10 that by adding any node to the set J, we get a lower anycast link cost.

Remark 4.3. Both the delivery probability and the transmission count ALC are *always-decreasing.*

We plot in Figure 4.6 the anycast forwarding gain $G(d_{iJ})$ for increasing size of candidate relay set. These plots assume that all links have equal delivery probability $p_{ij} = p$. For the delivery probability metric, the anycast link cost $(-\log p_{iC(i)})$ goes toward zero as $C(i)$ increases, and so the gain is unbounded. For the ETX metric, the anycast link cost goes toward 1 as $C(i)$ increases, giving a gain that is at most equal to $1/p$. The relative gain of anycast forwarding is therefore less sharp with the ETX metric than with the E2E metric.

4.4.2 Remaining Path Cost

RPC for unreliable links (E2E metric)

Since the E2E metric works with negative logarithms of probabilities, we must adapt equation (4.3) by converting the D_j back into the probability domain before composing them with other probabilities, and then convert the entire expression back into the logarithm domain:

$$\mathrm{E}\left[f(\mathbf{A})\right] = -\log\Bigg(q_{i1} e^{-D_1} + \sum_{j=2}^{n} \Bigg(\prod_{k=1}^{j-1} (1 - q_{ik}) \Bigg) q_{ij} e^{-D_j} \Bigg). \tag{4.7}$$

With unreliable links and no ARQ, we have $q_{ij} = p_{ij}$, and so the probability that at least one node in $C(i)$ receives the packet is $(1 - \prod_{k \in C(i)} \overline{p_{ik}})$. Substituting this probability and (4.7) into the remaining path cost (4.2) gives:

$$R_{iC(i)}^{best} = -\log\left(\frac{1}{1 - \prod_{k \in C(i)} \overline{p_{ik}}} \left(p_{i1} e^{-D_1} + \sum_{j=2}^{n} \left(\prod_{k=1}^{j-1} (1 - p_{ik}) \right) p_{ij} e^{-D_j} \right) \right), \tag{4.8}$$

and in the case of equal receiving probabilities (4.4) we have the somewhat more wieldy:

$$R_{iC(i)}^{best} = -\log\Bigg(\frac{p}{1 - (1 - p)^n} \sum_{j=1}^{n} (1 - p)^{j-1} e^{-D_j} \Bigg), \tag{4.9}$$

where we have used the superscript ^{best} to emphasize that the above expressions of remaining path cost are valid only for the policy ERS-best.

RPC for unreliable links with ARQ (ETX metric)

Computing q_{ij} appears more difficult as soon as we consider anycast forwarding using retransmissions, because the number of retransmissions is itself a random variable. However, recall that with ARQ the last transmission of the anycast forwarding phase takes place when any node $k \in C(i)$ receives the packet. Also the outcomes (i.e., which node(s) receive the packet) of each transmission are each independent, including the outcome of the last transmission. Therefore it is equivalent to consider the probabilities of events $\mathbf{A_j}$ in the last ARQ retransmission and the probabilities of events $\{\mathbf{A_j} | \cup_{\mathbf{k} \in \mathbf{C(i)}} \mathbf{A_k}\}$ for a single transmission, conditional on one node in $C(i)$ receiving a packet. Using (4.2) and (4.3), the expression of remaining path cost with the ETX metric is thus:

$$R_{iC(i)}^{best} = \frac{1}{1 - \prod_{k \in C(i)} \overline{p_{ik}}} \left(p_{i1} D_1 + \sum_{j=2}^{n} \left(\prod_{k=1}^{j-1} (1 - p_{ik}) \right) p_{ij} D_j \right), \tag{4.10}$$

with a similar simplification for the case of equal receiving probabilities:

$$R_{iC(i)}^{best} = \frac{p}{1 - (1-p)^n} \sum_{j=1}^{n} (1-p)^{j-1} D_j. \tag{4.11}$$

4.5 Anycast Forwarding with Asynchronous Duty-Cycling

We now return to the network model of asynchronous duty-cycled links that was defined in Chapter 3. As motivated in Section 3.4, we consider transmission energy costs, because with asynchronous duty-cycling the cost of sending a packet is inversely proportional to the duty cycle and is thus an obstacle to achieving extremely low duty cycles.

We have seen how anycast forwarding can improve performance over always-on unreliable links by increasing delivery probability or decreasing the number of retransmissions. It is clear that these improvements can be carried over to asynchronous duty-cycling (if the underlying channels are unreliable). Decreasing the number of retransmissions will reduce energy consumption and so is highly beneficial in this case too.

However, rather than just repeat the developments of the previous section in the context of duty-cycled links, we demonstrate here an entirely novel way of increasing energy efficiency through anycast forwarding that is specific to asynchronous duty-cycling. The gains that come from this technique can be significant, and apply equally well with reliable (as assumed here) as with unreliable links. If these underlying links are unreliable, then the gains from the technique presented in this section can be *cumulated* with the gains for unreliable links shown in the previous section.

An additional benefit is that the anycast forwarding gain of our technique increases not only with the size of the candidate relay set; it furthermore increases as the inverse of the duty cycle. In other words, the energy gains in comparison with unicast forwarding become more and more pronounced at the lowest duty-cycles, which are those where energy efficiency is most critical.

4.5.1 Anycast Link Cost

Section 2.4.3 showed that in the unicast case, an asynchronous duty-cycled link layer must employ preambles that are as long as the wakeup period, since a sending node does not know the next wakeup time of its neighbors. The cost of this wakeup preamble is inversely proportional to the duty cycle, and it becomes an obstacle to achieving extremely low duty cycles of less than 1%.

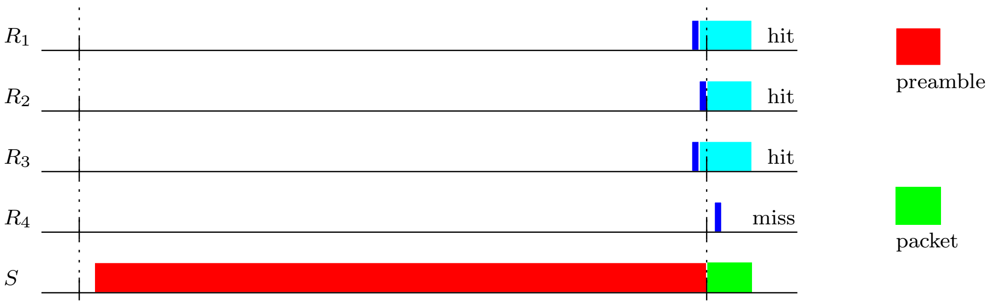
Can this cost be reduced when using anycast links? If this is the case, then anypath routing will compute routes that have lower overall cost than with single path; otherwise there will be no reduction in energy costs from anypath routing.

We generalize the definition of a preamble hit from Section 3.4.2 to be the event happening when *at least one* node in the set J wakes up during during the preamble transmission of a packet that is anycast to that set, and a preamble miss to be the complement of that event.

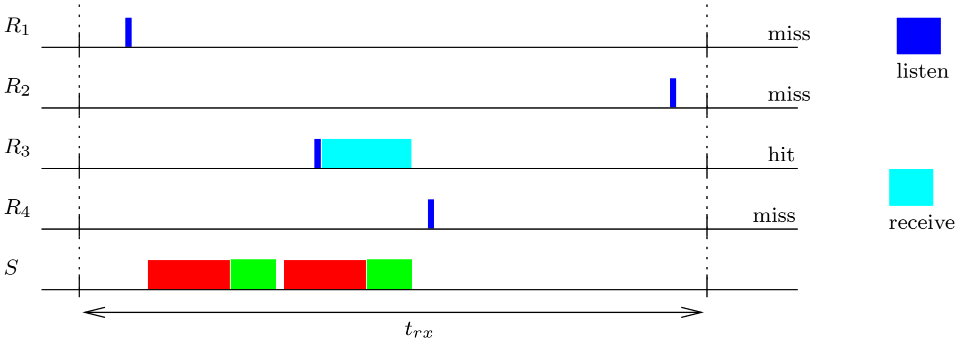
What are the energy costs in the anycast case? We can immediately note that without retransmissions, a node must transmit preambles of length t_{rx} in order to have a hitting probability equal to 1, irrespective of the size of the anycast set J. This is illustrated in Figure 4.7(a), where a sender S is transmitting to a candidate relay set $\{R_1, R_2, R_3, R_4\}$. This appears to be a pessimistic indication that there is no benefit to going from unicast to anycast transmission. However, the distribution of wakeup times in Figure 4.7(a) is atypical. Consider now Figure 4.7(b), where wakeup times are more uniformly distributed. The sender can now use a shorter preamble that is a fraction of the wakeup period t_{rx}. After sending the packet, the sender awaits an acknowledgement, and retransmits (possibly more than once) if no acknowledgement is received. In the figure, the sender must transmit twice in order to hit one of the nodes, but nonetheless the total transmission cost is smaller than for Figure 4.7(a). Beside the reduced transmission cost, this strategy has a second advantage that is significant in practice: a smaller number of nodes in the candidate relay set receive the packet than with the long preamble. This reduces the burden on the relay arbitration protocol and simplifies its implementation.

Having exposed the intuition behind anycast transmission with asynchronous duty-cycling, the remainder of this section is devoted to examining whether and how this observation can be exploited to reduce the energy cost of forwarding packets with anycast. Let us start off with the probability of preamble hit. We assume that receiver wakeup times are uniformly distributed in the interval t_{rx}, and that the duty cycle is low (i.e., $t_l/t_{rx} < 0.05$). Then the probability of preamble hit to an anycast set J, with a preamble of length $t_{preamble} = \lambda t_{rx}$ $(0 < \lambda < 1)$, is:

$$p_{hit} = 1 - (1 - \lambda)^n, \tag{4.12}$$



(a) $t_{preamble} \simeq t_{rx}$ ($\lambda \simeq 1$)



(b) $t_{preamble} < t_{rx}$ ($\lambda < 1$)

Figure 4.7: For anycast transmission, reducing preamble length and using ARQ is an efficient strategy with asynchronous duty cycling. Top: Without retransmissions, the sender S must use a long preamble to hit a node in the candidate relay set with high probability. Bottom: With retransmissions, a shorter preamble can be used for an overall reduction in transmission cost.

where $n = |J|$.

Recall from Section 3.4.2 that in the unicast case, we had the linear relation $p_{hit} = \lambda$, which is the special case of (4.12) for $n = 1$. This equation confirms the intuition of Figure 4.7, namely that the probability of anycast preamble hit for a set of size n increases *super-linearly* with the preamble length λ when $n > 1$. Further more the rate of increase is greater for larger values of n. In contrast, the probability of preamble hit for a single node j increases only *linearly* with λ when $n = 1$.

We plot p_{hit} in Figure 4.8, for varying λ and candidate relay set sizes n ($n = |C(i)|$). The left plot illustrates the previous observation: p_{hit} increases linearly with λ for $n = 1$, but super-linearly for greater values of n.

Equation (4.12) shows that p_{hit} depends on n as well as on λ. Here, the dependency is again favorable to anycast routing. The right plot of Figure 4.8 shows that for $n = 1$, we have $p_{hit} = \lambda$, whereas for $n > 1$, we have $p_{hit} > \lambda$. In other words, it becomes less

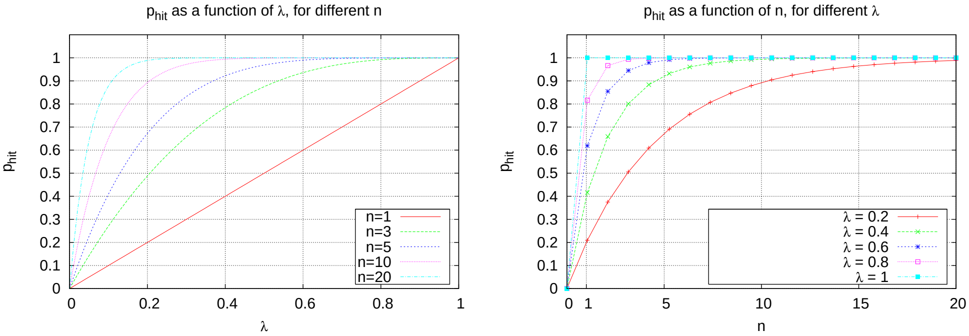


Figure 4.8: Probability of preamble hit p_{hit} for anycast forwarding with asynchronous duty-cycling. Left: p_{hit} for varying preamble length λ ($0 < \lambda < 1$). Right: p_{hit} for varying candidate relay set size n ($n = |C(i)|$).

and less expensive to hit *one* candidate relay as the number of candidates increases, as shown in the right-hand side plot.

Transmission-count metric

We do not define a delivery probability metric based on p_{hit}, because its interpretation would be unclear due to the dependence on (and tradeoff with) λ. What would the best operating regime be: to have long preambles and a high hitting probability, or shorter preambles but a reduced hitting probability? If our only aim is to increase delivery probability, then we should of course set $\lambda = 1$ and send full preambles. But the tradeoff is more subtle if we consider energy costs.

We now bring in the use of link-layer retransmissions, and consider a transmission-count metric. Recall that we are assuming reliable links, and so link-layer retransmissions are only necessary in case of a preamble miss. So, the expected number of transmissions is simply $1/p_{hit}$. Noting $n = |J|$, the expected number of transmissions for the duty-cycled link model is:

$$d_{iJ}(\lambda) = \frac{1}{p_{hit}} = \frac{1}{1-(1-\lambda)^n},$$

where we have emphasized with the notation $d_{iJ}(\lambda)$ the dependence on the free variable $\lambda \in [0, 1]$. λ is the preamble length, normalized to the wakeup period t_{rx}.

Similarly to the anycast preamble hit probability, the transmission-count ALC is minimized for $\lambda = 1$, and it does not take into account the energy required for packet transmissions. However, it shall be useful to establish the energy cost ALC metric in the next paragraph.

Energy metric

The *expected energy* required to anycast a packet to a set J, with asynchronous duty-cycling and link-layer retransmissions is the energy cost of a transmission multiplied by the expected number of transmissions:

Definition 4.11 (Energy transmission anycast link cost)**.** *The energy ALC for duty-cycled links with link-layer retransmissions is*

$$d_{iJ} = \min_{t_{preamble} \in [0, t_{rx}]} \frac{t_{preamble} + t_{pkt}}{p_{hit}} = \min_{\lambda \in [0,1]} \frac{\lambda t_{rx} + t_{pkt}}{1 - (1-\lambda)^n}. \tag{4.13}$$

A few comments are in order. Similar to the energy cost d_{ij} that we discussed in Section 3.4.2 for the unicast case, the anycast link energy cost varies as a function of λ. More interestingly, it is also dependent on n, the number of relay candidates. As expected, for $n = 1$ equation (4.13) reduces to the unicast forwarding cost of equation (3.1).

Let us now consider the general case. First, we note that for fixed λ, $d_{iJ}(\lambda)$ is monotonically decreasing with n.

Remark 4.4. The energy-cost ALC for asynchronous duty cycling is *always-decreasing*.

This is good news, since it means that the energy cost to reliably forward a packet will decrease as we grow the size of a candidate relay set. In fact,

$$\lim_{n \to \infty} d_{iJ}(\lambda) = \lambda t_{rx} + t_{pkt}. \tag{4.14}$$

Observe that λ is a free parameter. Therefore, as the size n of the candidate relay set grows, we can decrease λ toward the limit of 0, and have a forwarding cost approaching the cost of transmitting the packet alone, without the preamble. While this asymptotic trend points in the right direction, network density is bounded in real networks, and by consequence so will be the size of the candidate relay set. We must therefore look at the energy cost for finite values of n.

Now, let us fix n and look at $d_{iJ}(\lambda)$ as λ approaches 0 and 1. We have:

$$\lim_{\lambda \to 0} d_{iJ}(\lambda) = \infty, \tag{4.15}$$

and

$$\lim_{\lambda \to 1} d_{iJ}(\lambda) = t_{rx} + t_{pkt}. \tag{4.16}$$

Is the minimum reached for $\lambda = 1$, in which case the advantages of anycast forwarding are not materialized for finite values of n, or is there a λ for which the cost

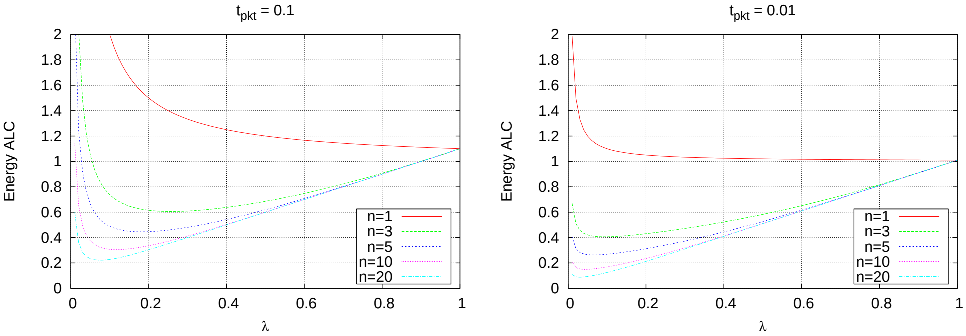


Figure 4.9: Swoosh! Anycast link cost for transmission energy as a function of preamble length λ, for different packet sizes t_{pkt} (relative to a normalized wakeup period $t_{rx} = 1$).

is lower? Answering this question analytically in the general case (i.e., for any n) is difficult, because finding the zeroes of the derivative (in λ) of (4.13) requires finding the zeroes of an order-n polynomial. We therefore compute it numerically. For the following paragraphs we shall therefore make statements which are not formal, but based instead on visual inspection of plots, and are aided by the fact that (4.13) is a fairly regular *"swoosh"* curve.

The first set of plots is in Figure 4.9. Each plot shows the anycast link cost for a different packet size t_{pkt}, where t_{pkt} is relative to the wakeup period t_{rx}. (In other words, we have normalized the wakeup period, $t_{rx} = 1$). Our observations about these plots are summarized as follows:

1. For $n = 1$, energy cost is minimized for $\lambda = 1$, giving $\lambda + t_{pkt}$. This corresponds to the unicast case of Section 3.4.2.

2. For $n > 1$, energy cost is minimized for a value $\lambda < 1$, and the minimum is less than $\lambda + t_{pkt}$. Furthermore, the minimum decreases with increasing n. This means that anycast transmission cost becomes cheaper and cheaper as the anycast set is expanded.

3. The energy cost increases very rapidly (towards ∞) on the left of its minimum. It increases less rapidly (towards $\lambda t_{rx} + t_{pkt}$) on the right of the minimum. This means that it is safer to be conservative and err on the side of choosing a λ that is greater than the one minimizing the energy cost.

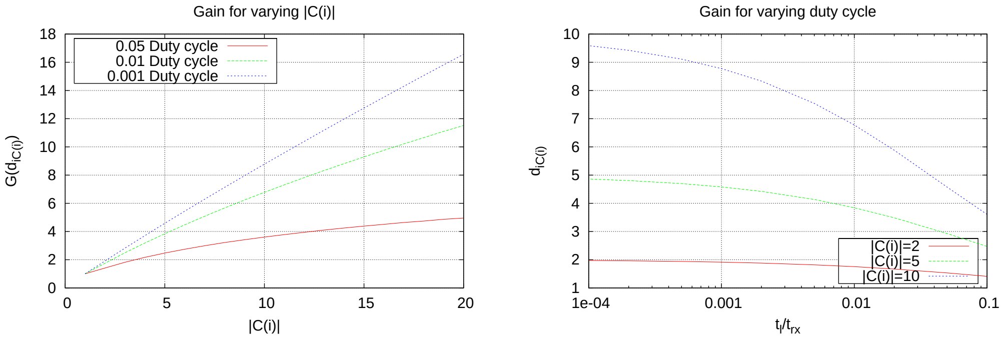
In the previous developments we expressed $d_{iC(i)}$ as a function of t_{pkt}, in order to highlight that the optimal λ depended on the length of a packet relative to a fixed listening interval t_{rx}.

We now turn to a more natural and interesting way of viewing things, namely that the packet length is fixed at $t_{pkt} = 1$, and t_{rx} increases. In other words, the true parameter that we are interested in is the duty cycle, since that is what a system designer would like to reduce as far as possible.

In Figure 4.10, we show the anycast forwarding gain for asynchronous duty cycling. In both plots, we have numerically computed the optimal λ for a given set of parameters.

Figure 4.10(a) shows the evolution of $G_{d_{iC(i)}}$ for increasing $|C(i)|$. It shows empirically that this anycast forwarding cost is always-decreasing; this cost converges for large n toward t_{pkt} and so the gain relative to unicast duty-cycling converges toward $\frac{t_{pkt}+t_{rx}}{t_{pkt}}$. We see that for a relatively high duty cycle of 0.05, we can reduce transmission cost by a factor of 2.5 with 5 candidate relays, and a factor of 5 with a large number of relays ($|C(i)| = 20$). This plot also suggests that gains are stronger for lower duty cycles; we show this in Figure 4.10(b).

These numbers are encouraging, in particular since they show that potential gains are increased *both* as we reduce the duty cycle *and* as we enlarge the number of nodes in $C(i)$. The challenge shall be of course to realize these gains when going from a theoretical model to a real, working system in Chapter 7.



(a) (b)

Figure 4.10: Anycast forwarding gain $G_{d_{iC(i)}}$ in transmission energy cost for asynchronous duty cycling. (a) Gain as a function of candidate relay set size $|C(i)|$. (b) Gain as a function of duty cycle $\frac{t_l}{t_{rx}}$.

4.5.2 Remaining Path Cost

To compute the remaining path cost, note that at each (re-)transmission, the probability of any node in $C(i)$ receiving the packet is the p_{hit} that we obtained in (4.12).

We have a similar situation as for always-on networks with ARQ: it is equivalent to consider a single transmission phase, conditional on the event that at least one node in $C(i)$ receives the packet. Thus, $R_{iC(i)}$ is obtained similarly as in (4.11):

$$R_{iC(i)} = \frac{\lambda_{opt}}{1-(1-\lambda_{opt})^n} \sum_{j=1}^{n} (1-\lambda_{opt})^{j-1} D_j \tag{4.17}$$

where λ_{opt} is the argument minimizing (4.13). We therefore cannot find a closed-form solution for $R_{iC(i)}$ in this case.

Note that unlike for the two always-on models discussed in this chapter, the remaining path cost here does not depend on the sender i. This is due to our modelling decision of having filtered (or reliable) links for this particular model. In this case, the relative probabilities of each node in J receiving a packet depend only on λ, and this λ itself depends only on $|C(i)|$. The fact that each node in $C(i)$ is equally likely to receive the packet at the end of the anycast forwarding phase will allow a vastly reduced algorithm complexity for CRS selection, and will simplify things for the real protocol design of Chapter 7.

4.6 Anycast Forwarding with Synchronous Duty Cycling

We now consider the two forms of synchronous duty-cycled links that were defined in Chapter 3, and see if and how anycast forwarding can be beneficial with these link models. Recall that we defined our cost metric for these schemes to be the *expected delay* to transmit a packet across a link, giving $d_{ij} = t_{rx}/2$. We chose this latency-based metric because synchronous duty cycling trades off energy efficiency for latency, and we are interested in the trade off that occurs as the duty cycle is reduced.

4.6.1 Anycast Link Cost

We have by now seen that anycasting a packet to a set of candidate relays has lower cost than sending to a single next hop. Specifically, anycasting can reduce expected transmission count, delivery probability, or energy cost to reach a node in a candidate relay set. Are such reductions also possible for the latency metric that we consider with synchronous duty cycling? Let us first examine the case of clustered duty cycling.

Clustered duty cycling

Consider a node i with a packet to send to *any* node in candidate relay set J. Does the expected latency change as a function of $|J|$? The answer is negative, because all nodes wake up for the *same* active period. If i is in the active period when it gets a packet to send, then it can send it immediately, and if i is not in the active period, then it must wait until the beginning of the next one to transmit. In either case, the fact that i can send to any node from J does not impact the latency.

Definition 4.12 (Latency ALC for clustered duty cycling)**.** *The latency ALC for duty-cycled links with clustered synchronization is*

$$d_{iJ} = t_{pkt} + \frac{t_{rx}}{2}$$

This case constitutes our first example of a link cost metric that *does not change with the size of the CRS J*. In fact, this ALC is *unicast-equivalent*, meaning that with clustered duty cycling, anycast forwarding cannot reduce the latency of a transmission over unicast forwarding.

The above conclusion is made under the assumption of *ideal* clustering, whereby all nodes share a single active period. In practice however, experiments with live networks [66] have shown that multiple clusters frequently do emerge. In such a situation, latency *can* be reduced by using anycast forwarding, at least for the border nodes that are part of multiple clusters. We do not explore further the performance gains of anypath routing with multiple clusters, because these gains are in a sense dependent upon the extent to which an underlying protocol performs erroneously.

As we have already pointed out (in Chapter 3), other choices of link metrics for clustered duty cycling are possible, for example a transmission-count metric. With such a metric (and assuming unreliable links), we would indeed see gains from anypath routing with clustered duty cycling. We do not consider this specific case because those gains are independent of the duty cycle. Furthermore the performance of anypath routing in reducing path transmission counts is examined in the case of always-on networks, and the results seen there can be interpreted as a proxy for the clustered duty cycle case.

Individual duty cycling

The situation changes in the case of individual duty cycling. Here, nodes also periodically (with period t_{rx}) listen for a duration of t_l. However, rather than having all nodes wake up according to a common schedule, each node has its own schedule and places its active period within the interval t_{rx} at an offset that it chooses individually. Nodes advertise their offset so that a sender knows when its intended receiver will awaken, and can schedule its transmission for that time.

Let us assume that active periods are uniformly distributed within the interval t_{rx}. What is the latency for a node i to send a packet to *any* node in the candidate relay set $C(i)$? To answer this question, we must clarify how the forwarding phase should work with individual duty cycling. Since i knows the schedules of each node in $C(i)$, it can select the effective relay that it will use for each packet based on these schedules. In order to minimize the delay, i simply chooses the node that next has its active period. The delay for node i to transmit to $C(i)$ is therefore the *minimum* of $|C(i)|$ random variables that are uniformly distributed over $[0, t_{rx}]$. This minimum can be computed using *order statistics* [38]. Define $\mathbf{X_{n,k}}$ to be the k-th smallest random variable out of n independent realizations of a random variable $\mathbf{X}$. We can show that if $\mathbf{X}$ is uniformly distributed over $[0, 1]$, then $\mathbf{X_{n,k}}$ has beta distribution with parameters k and $n - k + 1$. The mean of the beta distribution with parameters α and β is $\frac{\alpha}{\alpha+\beta}$, and in the present case we have $k = 1$ and $n = |C(i)|$, giving a mean of $\frac{1}{n+1}$.

Definition 4.13 (Latency ALC for individual duty cycling)**.** *The latency ALC for duty-cycled links with individual synchronization is*

$$t_{pkt} + \frac{t_{rx}}{|C(i)| + 1}$$

This anycast link cost decreases as the inverse of $|C(i)|$; it is therefore *always-decreasing.*

Sender-driven forwarding. In the previous models for always-on networks and for asynchronous duty cycling, we saw that the anycast forwarding was *receiver-driven*: a sender anycasting a packet to $C(i)$ cannot decide *before* transmission to which node it would like to send a packet, because there is uncertainty about which nodes will receive the packet, and this uncertainty is only resolved *after* the transmission.

Unlike for all the link models seen so far, anycast forwarding with individual duty cycling is however *sender-driven*: the sender knows the schedules of the candidate relays, and can select before transmission which node in $C(i)$ it will send to.

4.6.2 Remaining Path Cost

For clustered duty cycling, the ALC is unicast-equivalent, and anypath routing will never select a candidate relay set containing more than one node. As such, the remaining path cost is simply the distance of the next hop to the destination, as in single-path routing.

We now turn to individual duty cycling, where we have seen that a forwarding node with candidate relay set J sends its packet to the node in J with the next active period. The remaining path cost for each packet therefore depends on the offset in the interval t_{rx} at which it is generated (by an upper layer) or is received for forwarding. Assuming that the phase of packet arrivals in the interval t_{rx} is random and uniformly distributed, then over the long-term we will use each node J with equal probability.

Definition 4.14. *With individual synchronized duty cycling, the remaining path cost when using J as candidate relay set is*

$$R_{iJ} = \frac{1}{|J|} \sum_{j=1}^{|J|} D_j.$$

A couple of comments on this definition. First, the remaining path cost is independent of the sender i, because the relative probabilities of using each node in J as the effective next-hop relay depend only on the size of J, and not on the sending node i. A second comment is that the RPC here does not take into account the ERS-best policy in any way (and hence its expression is much simpler than those seen previously): since the sender chooses the relay before sending, there is no need to arbitrate between multiple receiving nodes, and so the ERS policy is never invoked.

Note that it may happen that the active periods of two nodes in J happen simultaneously. In such a case, there would be a requirement for an ERS policy to select which node in J to use as a relay, and we would expect the use of ERS-best as for all previous link models. Our definition of RPC above therefore slightly overestimates R_{iJ}, because it assumes that such ties are resolved at random[5]. The degree of imprecision depends on how often nodes have overlapping active periods. In the cases of interest, namely with duty cycles of 0.01 and lower, the probability of such overlaps is low, and so our approximation suffices.

4.7 Summary

This chapter has shown that anycast forwarding is a practical and general way to exploit spatial diversity for point-to-point communication across multiple wireless hops.

Specifically, we have defined a framework for anypath routing that includes the notions of anycast link cost and remaining path cost, distinguishes between sender-driven and receiver-driven anycast forwarding, and shows that anycast forwarding is applicable both with underlying broadcast and with underlying unicast links. We introduced two new link layer mechanisms that show how anycast forwarding can increase energy efficiency or decrease latency with low-power duty-cycled links. Table

---

[5] This corresponds to the ERS-all policy that is defined in Chapter 6.

Link model	**Metric Type**	**Sender-driven or Receiver-driven**	**Metric Interpretation**
Always-on unreliable links	Delivery probability	Receiver	Probability that any node in $C(i)$ receives packet after a single transmission (4.5).
Always-on unreliable links with ARQ	Transmission count	Receiver	Expected number of transmissions until any node in $C(i)$ receives packet (4.6).
Asynchronous duty cycled links with ARQ	Transmission energy	Receiver	Expected energy cost to send a packet to any node in $C(i)$ (4.13).
Clustered duty cycling (synchronous)	Latency	Sender	Expected delay to send a packet to any node in $C(i)$.
Individual duty cycling (synchronous)	Latency	Sender	Expected delay to send a packet to any node in $C(i)$.

Table 4.1: Summary of anycast forwarding applied to different link models.

4.1 summarizes the different anycast forwarding techniques and their properties, and Table 4.2 gives their costs.

Link model	**Link Cost** d_{ij}	**Anycast Link Cost** $d_{iC(i)}$	**Type**
Always-on unreliable links	$-\log p_{ij}$	$-\log(1-\prod_{j\in C(i)}\overline{p_{ij}})$	Always-decreasing
Always-on unreliable links with ARQ	$\frac{1}{p_{ij}}$	$\frac{1}{(1-\prod_{j\in C(i)}\overline{p_{ij}})}$	Always-decreasing
Asynchronous duty cycled links with ARQ	$t_{rx}+t_{pkt}$	$\min_{\lambda\in[0,1]}\frac{\lambda t_{rx}+t_{pkt}}{1-(1-\lambda)^n}$	Always-decreasing
Clustered duty cycling (synchronous)	$t_{rx}/2$	$t_{rx}/2$	Unicast-equivalent
Individual duty cycling (synchronous)	$t_{rx}/2$	$\frac{t_{rx}}{\lvert C(i)\rvert+1}$	Always-decreasing

Table 4.2: Summary of anycast link costs, their properties, and comparison with unicast link costs.

Chapter 5

Anypath Routing

> *He who wanders is not always lost.*
>
> - J.R.R. Tolkien

Notation and Key Concepts

Physical cost model	Anycast link cost where adding a node j to $C(i)$ that is further than i from destination always increases the distance from i to the destination.
$\mathcal{R}$	Anypath route Directed graph where every node (but the source) is a successor of the source, and every node (but the destination) is an predecessor of the destination.
$Cost(\mathcal{R})$	Anypath route cost Average cost to send a packet across an anypath route.
Shortest anypath route	Route $\mathcal{R}$ with lowest $Cost(\mathcal{R})$ out of all anypath routes between given source and destination. Is always *acyclic* if cost model is physical.
The destination node is by convention node 1 throughout this chapter	

The purpose of an anypath routing algorithm is to determine the candidate relay set (for a given destination) at each node. How should the algorithm assign $C(i)$ to each node i? Recall that the anycast link costs seen in Chapter 4 decrease monotonically with growing size of the candidate relay set (they are *always-decreasing*). Based on this observation alone, one would set the candidate relay set at each node to be as large as possible in order to minimize transmission costs, i.e. set $C(i) = N(i)$. But this would result in packets being anycast randomly from neighbor to neighbor, with no guarantee of making progress toward the destination.

A second strategy would be to take the largest possible $C(i)$ at each node, under the constraint that no loops should arise in the resulting topology. While this strategy appears to be sensible, we shall see that it is doubly off the mark. One reason is that the best candidate relay set is not always the largest one. The other reason is certainly the more startling: *allowing loops turns out in certain cases to be beneficial!*

A third strategy would be to compute the shortest-path distances from every node to the destination using a standard single-path algorithm, and set $C(i)$ to be the set of all nodes with lower single-path distances than i. This approach is the one taken by Morris et al in XOR [9]. It also appears to be a sensible one, since shortest single-path distance is *the* prototypical notion of distance in a multi-hop network; making a packet advance to any node that is closer to the destination than the previous relay can hardly seem to be a counter-productive action.

However, as we shall see, there are both situations where *not* placing a node with lower single-path distance in the CRS improves routing performance, and situations where placing a node with *greater* single-path distance improves routing performance. The main reason why the XOR approach based on single-path distances does not lead to the best candidate relay sets is that link costs with anycast forwarding are not equal to unicast link costs, and so shortest single-path costs are not a good estimator of the cost to forward a packet using anycast forwarding.

The goal of this chapter is to build upon the concepts of anycast forwarding developed previously and to formulate and prove a correct strategy to select "good" candidate relay sets at each node. This requires us to define the notion of *anypath route*, that we have only informally used so far. We also define the *cost* of an anypath route; once this notion of cost is established it becomes meaningful to talk of *shortest* anypath routes. After this, we introduce an algorithm to compute the shortest anypath routes. This algorithm is a generalization of the Bellman-Ford algorithm for single-path routing. We prove its convergence and correctness, as well as some interesting properties of shortest anypath routes.

The first step, however, is to return to the definitions of anycast link cost and remaining path cost from the previous chapter, and introduce the notion of a *physical* cost model, that shall be needed to distinguish important cases of anypath routing.

Summary of contributions

The concept of shortest-path routing is pervasive throughout networking theory and practice. This chapter brings together shortest-path routing and anycast forwarding, and shows how the mechanism of anycast forwarding can be integrated into a routing framework that we call *shortest anypath routing*. Specifically, the contributions of this chapter include:

- A definition of anypath routes and their costs, as well as an algorithm to compute these costs,

- the distinction between a physical cost model, for which the shortest anypath route has no cycles, and a non-physical cost model, where the shortest anypath route may have cycles (!), and
- a distributed algorithm to compute shortest anypath routes that has the same upper bound on convergence time as the Bellman-Ford algorithm for computing single-path routes.

5.1 Physical Cost Models

The benefit of anycast forwarding is that in many cost models, it is less costly for a node to transmit to any one of a set of neighbors than to a single neighbor.

Should we be interested in *any* anycast link cost (ALC) metric that is always-decreasing or sometimes-decreasing? While the answer to this question is positive, it turns out that we must make a distinction that is both of technical and semantic importance to design and reason about anypath routing.

The intuition is the following: consider a node i with a given candidate relay set J and a cost D_i to the destination. Recall from the previous chapter that the cost to reach the destination from a node with CRS J is the sum of anycast link cost and remaining path cost: $D_i = d_{iJ} + R_{iJ}$[1]. Consider now a neighbor k of node i with distance $D_k > D_i$. Can node i decrease its cost to the destination by adding node k to its candidate relay set?

If the answer to this question is positive, then we have following highly counterintuitive situation: *we can reduce the cost of sending a packet from i to the destination by going through a relay that has a higher cost to the destination than i itself!* This apparent paradox motivates the definition of a *physical* cost criterion, which is a technical condition on the ALC and RPC under which such counter-intuitive situations are impossible.

Definition 5.1 (Physical cost model)**.** *Consider a node i with candidate relay set J. The cost to reach the destination from i is $D_i = d_{iJ} + R_{iJ}$. Let $k \in N(i) \setminus J$ be a neighbor of i that is not in J, and for which $D_k \geq D_i$. Define the set J' to be the union of i's candidate relay set with k: $J' = J \cup k$.*
A physical cost model *is one for which*

$$d_{iJ'} + R_{iJ'} \geq d_{iJ} + R_{iJ},$$

---

[1] Of course computing R_{iJ} requires knowing the distances D_k from each node $k \in C(i)$ to the destination, which we do not yet know how to compute; we can do without it for the present section.

for all possible combinations of i, J, *and* k.

Note that a unicast-equivalent link cost is trivially a physical cost (Property 4.5).

Analogy with single-path routing

Informally, saying that a cost metric is physical is equivalent to saying that going through a node that is further from the destination than ourself can only *increase* our cost to the destination. Is there a similar notion in the single-path sense? A comparable requirement to our physical cost model is to require in the single-path case that $D_i > D_k$. In other words, if node i uses as next-hop node k, then i is *further* from the destination than k.

The requirement that a node is further from the destination than the next hop in a path is so obvious that it is often left unstated in the context of single-path routing. But it is a key requirement for many graph algorithms [7], and it is expressed in a more direct fashion by saying that *link distances are always positive.* Indeed, if the link distance between node i and k is negative, then node i may perfectly be closer to the destination than its next-hop node k. A physical cost model in anycast forwarding can thus be seen as conceptually similar to a link model in unicast that excludes negative link costs. Recall that we have excluded negative link distances right from the outset of Chapter 3; non-physical costs of anycast links are possible even when all individual links costs are positive.

5.1.1 Always-on Networks

We now return to the ALCs defined in Chapter 4. We start with always-on networks, and show that the expected transmission count (ETX) anycast link cost is a physical model, but that the end-to-end packet delivery (E2E) anycast link cost is not.

Proposition 5.1. *The expected transmission count (ETX) anycast link cost is a physical cost model.*

Proof. To lighten notation, we write p_k instead of p_{ik} in the following, given that we are only considering links from i to a neighbor. Consider a node i with candidate relay set $J = \{j\}$. Under the ETX metric and under the ERS-best model, the cost D_i is determined by (4.6) and (4.10), and is $D_i = \frac{1}{p_j} + D_j$. Now add a node k with $D_k \geq D_i$ to the candidate relay set, to obtain a set $J' = \{j, k\}$, and examine the distance $d_{iJ'} + R_{iJ'}$. From (4.6) we have the anycast link cost from i to J':

$$d_{iJ'} = \frac{1}{1 - (1 - p_j)(1 - p_k)},$$

and from (4.10) we have the remaining path cost:

$$R_{iJ'} = \frac{1}{1-(1-p_j)(1-p_k)}(p_j D_j + (1-p_j)p_k D_k),$$

where we have used the hypothesis that $D_k < D_j$.

We now consider the sum of these two quantities and show that it is greater than D_i:

$$\begin{aligned} d_{iJ'} + R_{iJ'} &= \frac{1}{1-(1-p_j)(1-p_k)} + \frac{1}{1-(1-p_j)(1-p_k)}(p_j D_j + (1-p_j)p_k D_k) \\ &= f(p_k), \qquad p_k \in [0,1] \\ &\overset{(a)}{\geq} f(0) \\ &= \frac{1}{p_j} + D_j \\ &\overset{(b)}{=} d_{iJ} + R_{iJ} = D_i, \end{aligned}$$

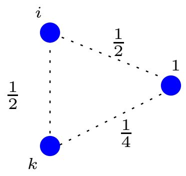
where we used in (a) the fact that $0 < p_k < 1$, and the quantities p_j, $(1-p_j)$, and D_j are positive, and in (b) the definition of D_i.

We have not yet shown that the expected transmission count ALC is a physical cost model, since the development above considers only the special case where $|J| = 1$. Let us now consider the general case where the candidate relay set has an arbitrary number of nodes: $|J| = n$. Consider now another candidate relay set H consisting of a single node h, such that $p_h = p_{iJ} = (1 - \prod_{j \in J}(1-p_{ij}))$, and $D_h = R_{iJ}$ (where R_{iJ} is computed according to (4.10)). Observe that $d_{iJ} = d_{iH}$ and $R_{iJ} = R_{iH}$, and that it is equivalent to consider either the candidate relay set J or our constructed set H. Therefore, any general case J can be reduced to a singleton CRS, a situation for which the above development has shown that the physical cost property holds.

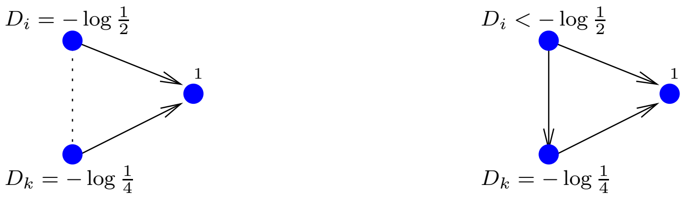
Finally, we have only considered the ERS-best policy in the above development. Note that with any other policy, the remaining path cost when adding node k is greater than with ERS-best, and so the fact that the expected transmission count ALC is a physical cost metric for ERS-best implies that it is a physical cost metric for any ERS. ☐

Proposition 5.2. *The end-to-end delivery probability (E2E) anycast link cost is not a physical cost model.*

Proof. We prove this using a counter-example. Consider the network of Figure 5.1, where node 1 is the destination. The link from i to the destination has delivery



(a) Network with source i and destination 0.



(b) $D_i < D_k$

(c) D_i decreases!

Figure 5.1: The packet delivery probability ALC is not a physical cost model. Link delivery probabilities are annotated in (a). The distance from k to the destination *decreases* in (c) if it adds as candidate relay a node k which in (b) was actually *further* than itself from the destination!

probability $p_{i1} = \frac{1}{2}$, the link from k to the destination has delivery probability $p_{k1} = \frac{1}{4}$, and the link between i and k has probability $p_{ik} = \frac{1}{2}$. In Figure 5.1(b), node i has as candidate relay set the singleton set containing the destination only, i.e., $J = \{1\}$. The remaining path cost is therefore $R_{iJ} = 0$, and we have $D_i = d_{iJ} + R_{iJ} = -\log\frac{1}{2}$. Node k has the same candidate relay set consisting of the destination only, giving us $D_k = -\log\frac{1}{4} > D_i$.

Consider now Figure 5.1(c), where node i uses as CRS the set $J' = \{1, k\}$. If the delivery probability ALC were a physical cost model, the cost at node i would increase with this candidate relay set J', because $D_k > D_i$. More precisely, we should have $d_{iJ'} + R_{iJ'} \geq d_{iJ} + R_{iJ}$, where $J = \{1\}$. However, using equations (4.5) and (4.8) we can compute that

$$d_{iJ'} = -\log\left(1 - (1 - p_{i1})(1 - p_{ik})\right) = -\log\frac{3}{4},$$

and

$$R_{iJ'} = -\log\left(\left(1 - (1 - p_{i1})(1 - p_{ik})\right)\left(p_{i1} + (1 - p_{i1})p_{ik}p_{k1}\right)\right) = -\log\frac{3}{4}.$$

Putting both together, we obtain

$$d_{iJ'} + R_{iJ'} = -\log\frac{3}{4} - \log\frac{3}{4} < -\log\frac{1}{2} = d_{iJ} + R_{iJ},$$

showing that the delivery probability ALC is not a physical cost model.

Since this example is simple, we can also directly compute the raw probabilities. A packet sent by i arrives successfully at the destination if either it is received directly by the destination, or it is successfully relayed by node k to the destination. The probability p of successful delivery is thus:

$$p = p_{i1} + (1 - p_{i1})\, p_{ik}\, p_{k1} = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{4} = \frac{9}{16}.$$

Of course this “direct” computation gives exactly the same result as the development above; we can verify that $e^{-(d_{iJ'} + R_{iJ'})} = \frac{9}{16}$. □

Interpretation

A first, direct interpretation of why the link delivery probability ALC is not a physical cost model is that the reduction in the anycast link cost when going from a candidate relay set $J = \{1\}$ to $J' = \{1, k\}$ is greater than the increase in the remaining path cost. Therefore, the sum D_i of anycast link cost and remaining path cost is lower with J' than with J.

This interpretation however is very close to the technical development. A more intuitive one is to recall that this example uses unreliable links without retransmissions, or equivalently, nodes without buffers. So, if node i transmits a packet that is not received by the destination, but that is received by node k, then allowing k as a relay as in Figure 5.1(b) will result in a higher delivery probability than if the packet were simply dropped. Of course the packet can still be lost, if neither the destination nor k receive i’s transmission, or if k (alone) receives it from i but the subsequent transmission from k to the destination is lost.

Of course, a probability is not a physical quantity, and in contrast, a physical cost model counts a *physical* quantity. For example, ETX counts the expected number of transmissions to reach the destination. Similarly, the ALCs introduced in the context of synchronous duty cycling (Section 4.6) count the average energy and the average latency necessary to send a packet to the destination.

5.1.2 Duty-Cycled Networks

Proposition 5.3. *The latency ALC for synchronous duty cycled links is a physical cost model.*

Proof. For clustered duty cycling, this follows trivially from the fact that the latency metric is in this case unicast-equivalent (Property 4.5).

For individual duty cycling, we consider a node i with candidate relay set $J = \{j\}$. Under the ALC and RPC defined for individual duty cycling, (Definitions 4.13 and 4.14), the cost D_i is $D_i = \frac{t_{rx}}{2} + D_j$. Now add a node k with $D_k \geq D_i$ to the candidate relay set, to obtain a set $J' = \{j, k\}$, and let us examine in the distance $d_{iJ'} + R_{iJ'}$. The anycast link cost $d_{iJ'}$ is

$$d_{iJ'} = \frac{t_{rx}}{3},$$

and the remaining path cost $R_{iJ'}$ is

$$R_{iJ'} = \frac{D_j + D_k}{2}$$

We now consider the sum of these two quantities and show that it is greater than D_i:

$$\begin{aligned} d_{iJ'} + R_{iJ'} &= \frac{t_{rx}}{3} + \frac{D_j + D_k}{2} \\ &\overset{(a)}{\geq} \frac{t_{rx}}{3} + \frac{D_j + D_i}{2} \\ &\overset{(b)}{=} \frac{t_{rx}}{3} + \frac{D_j + \frac{t_{rx}}{2} + D_j}{2} \\ &= \frac{7}{12} t_{rx} + D_j \\ &\overset{(c)}{>} D_i, \end{aligned}$$

where we used in (a) our starting assumption that $D_k \geq D_i$, and in (b) and (c) the definition of D_i.

This development considers only the case where $|J| = 1$; the proof is completed for the general case $|J| = n$ by the same argument as in Proposition 5.1. ☐

5.2 Shortest Anypath Routes

Before we can propose an algorithm for nodes to compute an anypath route we must first define the cost of an anypath route, and defining this cost requires first clarifying exactly what is meant by an anypath route. The anypath route is the union of all the directed anycast links $(i, C(i))$ between a source and a destination:

Definition 5.2 (Anypath route and acyclic anypath route)**.** *An* anypath route $\mathcal{R}$ *from a source to a destination is a directed graph where every node (but the source) is a successor of the source, and every node (but the destination) is a predecessor of the destination. An* acyclic *anypath route is an anypath route that contains no cycles.*

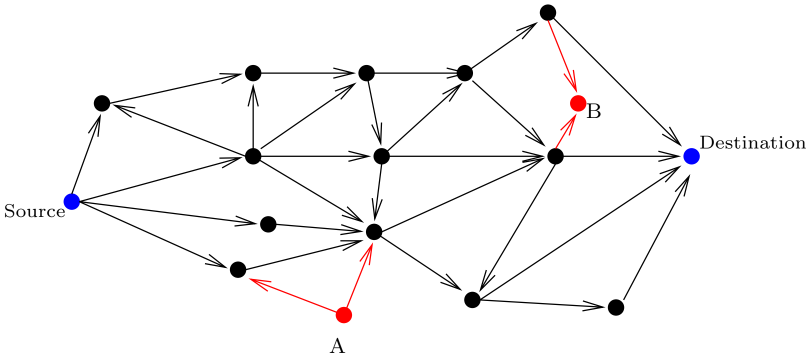


Figure 5.2: Illustration of anypath routes. The black nodes and links make up an anypath route between the source and destination. With the addition of either of the two red nodes (A and B) and their incident links, the graph ceases to be an anypath route (one node is not a successor of the source, one is not a predecessor of the destination). This anypath route is *acyclic*. Illustration of Example 5.1

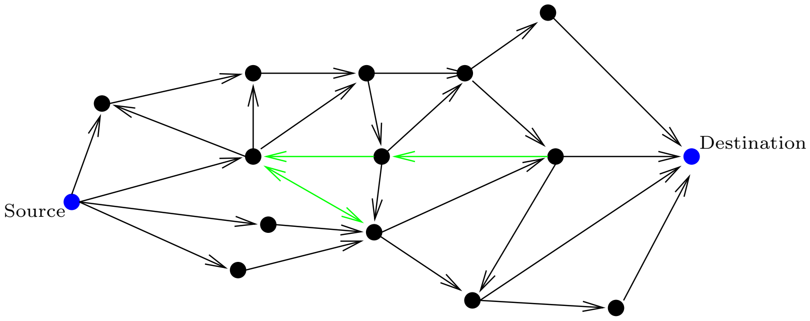


Figure 5.3: An anypath route may contain cycles. This route is similar to the route of Figure 5.2, but the modified links (in green) make it cyclic. It contains cycles of length two, three, four, five, six, and seven. Illustration of Example 5.1.

Example 5.1 (Cyclic and acyclic anypath routes)**.** The black nodes and links of Figure 5.2 make up an anypath route between the source and destination. By connecting node A as shown we cease to have an anypath route since A is not a successor of the source. Similarly, node B is not a predecessor of the destination, so by adding it we also cease to have an anypath route.

While the anypath route shown in Figure 5.2 is *acyclic*, our definition does not require that an anypath route have no cycles. An illustration of a cyclic anypath route is shown in Figure 5.3.

5.2.1 Cost of Anypath Routes

Having defined the notion of anypath route, we now need a definition for the *cost* of an anypath route, just as we require a notion of path cost in the single-path case. Note that we already touched upon the anypath cost in the previous section, where we considered a node i with a known candidate relay set, and established the anypath cost D_i to reach the destination from this node as the sum $d_{iJ} + R_{iJ}$ of the anycast link cost and remaining path cost. However the remaining path cost was itself defined in terms of the anypath costs $D_j, j \in C(i)$ from the nodes in the candidate relay set, and so this cannot suffice to globally define the cost of an anypath route.

Cost of a trajectory in an anypath route

At each use of an anypath route, a packet may traverse a different sequence of nodes to go from the source to the destination, since there are (in general) multiple candidate relays at each hop. We must therefore first define the cost of a traversal across a given sequence of nodes, before we can reason about the overall cost of the anypath route.

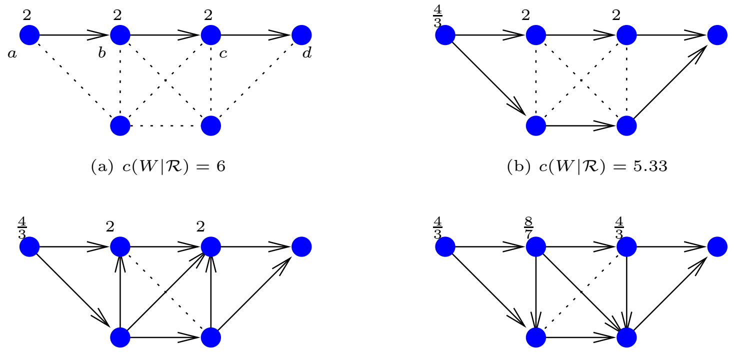
A trajectory T in an anypath route $\mathcal{R}$ is a subgraph of $\mathcal{R}$ that connects the source and the destination. Note that with the ERS-best policy considered so far, a trajectory will always be a simple path. With other ERS policies, and more specifically with those that allow multiple effective relays (such as ERS-all, that is introduced in the next chapter), a trajectory may contain branches. We can now define the cost of a trajectory *relative to the anypath it traverses* anypath:

Definition 5.3 (Cost of a trajectory in an anypath route)**.** *Let* $T = (s, n_1, n_2, \ldots n_k, 1)$ *be a trajectory in anypath route* $\mathcal{R}$*. The* cost *of* T relative to $\mathcal{R}$ *is the sum of the constituent anycast link costs in* $\mathcal{R}$ *of the nodes in the path* T*:*

$$c(T|\mathcal{R}) = \sum_{i \in T} d_{iC(i)} = d_{sC(s)} + d_{n_1C(n_1)} + d_{n_2C(n_2)} + \ldots d_{n_kC(n_k)}.$$

It is important to emphasize that the cost of T depends on the anypath route $\mathcal{R}$ that it traverses, as the following example will show. Indeed, the anycast link cost $d_{n_iC(n_i)}$ that goes into the computation of $c(T|\mathcal{R})$ above depends on the entire candidate relay set $C(n_i)$ and not only on the next hop n_{i+1} in the trajectory.

Example 5.2 (Cost of a trajectory depends on containing anypath)**.** We illustrate this dependence in Figure 5.4, where we are interested in computing the cost of the path $T = (a, b, c, d)$ relative to four containing anypath routes. In this network all links have delivery probability 0.5, and the cost metric is ETX (see Table 4.1). In case (a), the containing anypath is equal to T itself. The anycast link costs at nodes $a, b,$ and c are therefore each $(1 - 0.5)^{-1}$ and so the cost of T relative to $\mathcal{R}$ is 6. In case (b), the anycast link cost at node a is now $(1 - 0.5^2)^{-1} = 4/3$, and so the cost of T is 5.33. In case (c), the cost is the same as in (b), because the anycast link costs at nodes b and c are not affected by the additional incoming links. Finally in case (d) the cost of T relative to its containing anypath route is now $(1 - 0.5^2)^{-1} + (1 - 0.5^3)^{-1} + (1 - 0.5^2)^{-1} = 3.81$.



(a) $c(W|\mathcal{R}) = 6$ (b) $c(W|\mathcal{R}) = 5.33$

(c) $c(W|\mathcal{R}) = 5.33$ (d) $c(W|\mathcal{R}) = 3.81$

Cost of the same walk $W = (a, b, c, d)$, relative to four different anypath routes.

Figure 5.4: Cost of a trajectory relative to four different anypaths. The metric is ETX, and all links have delivery probability $1/2$. Illustration of Example 5.2.

Anypath route cost

Of course there are multiple possible trajectories for a packet to go from the source to the destination in an anypath route. Each possible trajectory T happens with some probability $P(T)$. If we view the effective trajectory of a packet as a realization, it is natural to define the cost of an anypath route as an expected cost:

Definition 5.4 (Anypath route cost)**.** *The cost of an anypath route $\mathcal{R}$ is the expected cost of a trajectory across that route.*

$$Cost(\mathcal{R}) = \sum_{T \in \mathcal{R}} P(T) \cdot c(T|\mathcal{R}),$$

where the sum is over all *possible trajectories from the source to the destination of $\mathcal{R}$.*

This definition corresponds to our intuition: the cost of an anypath route should correspond to the average cost of sending a packet across this route. A few remarks on the anypath route cost:

- The probability of a path T depends on the choice that is made by the ERS policy at each anycast forwarding phase. Thus, it depends on costs of the nodes in the route, since when multiple nodes receive a packet, the ERS policy chooses the effective relay based on their cost to the destination[2]. Since the cost of an anypath route depends on the costs from interior nodes in that route, we must therefore know the costs of the interior nodes in order to compute the cost of the route.
- If the anypath route consists of a single path, there is only one trajectory across it from the source to the destination, that is the route itself. Its cost is the sum of the constituent link costs. Therefore the anypath route cost generalizes the cost of a single path route.
- The anypath route cost is based on anycast link costs, that are used to compute the cost of each trajectory. However the remaining path cost is completely absent from the definition, and is not even used indirectly.

5.2.2 Shortest Anypath Routes

Now that we have a precise notion of cost for anypath routes, the *shortest anypath route* between two nodes is defined naturally as the anypath route with lowest cost among all possible anypath routes. Similarly, the shortest *acyclic* anypath route is the acyclic anypath route with lowest cost. Note that the shortest anypath route is not always unique: just as in single-path routing, there may be more than multiple shortest anypath routes with equal cost.

Definition 5.5. *The* shortest (acyclic) anypath route *from a source to a destination is the anypath route $\mathcal{R}$ with lowest cost among all (acyclic) anypath routes between*

---

[2] We shall see in the next chapter that this dependence is lifted for some other ERS policies.

the source and destination. The anypath distance *(or equivalently,* anypath cost*) between two nodes is the distance (or cost) of the shortest anypath route between the two nodes.*

Presence (or absence) of cycles in shortest anypath routes

It may appear surprising that we have allowed the presence of cycles in our definition of anypath routes. Returning to the definition of *physical* cost models (Definition 5.1), we can now show that in the important case of physical networks, a cyclic anypath route can never have lower cost than the shortest acyclic anypath route. This is fortunate, since reasoning about the subset of anypath routes that are acyclic is easier than reasoning about all possible anypath routes.

Proposition 5.4. *In a physical cost model, no cyclic anypath route can have lower cost than the shortest acyclic anypath route.*

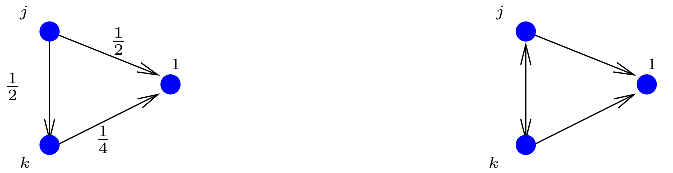
Proof. By contradiction. Assume that the shortest anypath route $\mathcal{R}$ between two nodes is not acyclic. It therefore contains at least one cycle. Let us consider one cycle $(n_1, n_2, n_3, \dots n_{k-1}, n_1)$, with nodes having destination distances $(D_1, D_2, D_3, \dots D_{k-1}, D_1)$. Since this is a cycle, one of two cases must be true: either $D_1 = D_2 = D_3 = \dots = D_{k-1}$, or there are two consecutive nodes (i, j) in the cycle with $D_i < D_j$.

We consider each of these two cases individually.

- **Case 1:** $D_1 = D_2 = D_3 = \dots = D_{k-1}$. No cycle can contain the destination, since the destination has no successor. Therefore any cycle in an anypath route cannot be *closed*; in other words there must be at least one node in $(n_1, n_2, n_3, \dots n_{k-1}, n_1)$ with a successor that is *not* in the cycle. Let us call this node n_i; n_i has one successor n_{i+1} in the cycle, and a set of other successors K (with $|K| \geq 1$). (In the terminology used previously this amounts to saying that $C(n_i) = \{n_{i+1}\} \cup K$). Now, if we consider the definition (Definition 5.1) of a physical cost model, and set $J = K$ and $J' = \{n_{i+1}\} \cup K$, we see that we can remove n_{i+1} from node n_i's successors (i.e., set $C(i) = K$) and reduce the cost D_i.

- **Case 2:** $D_i < D_j$ for two consecutive nodes (i, j) in the cycle. Since anycast link costs are positive, node i must have another successor(s) in addition to j (or else we would have $D_i = d_{i\{j\}} + R_{i\{j\}} = d_{i\{j\}} + D_j > D_j$). Again, considering the definition (Definition 5.1) of a physical cost model shows that the cost D_i can be reduced by removing j from node i's candidate relay set.

Returning to our shortest anypath route $\mathcal{R}$, let us construct the anypath $\mathcal{R}'$ by applying either of the two modifications above to every cycle in $\mathcal{R}$. Calling D'_k the



(a) Shortest *acyclic* anypath from j to 1.
$e^{-D_i} = \frac{9}{16}$

(b) Shortest anypath (cyclic)
$e^{-D_i} = \frac{18}{29} < \frac{9}{16}$

Figure 5.5: Shortest anypath route may contain cycles with a non-physical cost model. The cost model is delivery probability, and link delivery probabilities are annotated next to each link. The source and destination are nodes j and 1 respectively. The shortest acyclic anypath is shown in (a) and has lower delivery probability than the shortest anypath in (b), that contains a cycle. Illustration of Example 5.4.

distance from node k to the destination, we have that $D'_k \leq D_k$ for all nodes, with the inequality being sharp for at least one node (the one whose candidate relay set was pruned according to the modifications above).

In consequence, all possible trajectories have either lower or equal cost over $\mathcal{R}'$ than over $\mathcal{R}$, with at least one having a lower cost (ie, a trajectory going through the node whose candidate relay set was pruned). The constructed route $\mathcal{R}'$ has lower cost than $\mathcal{R}$, contradicting our initial assumption. $\square$

Corollary 5.5. *A routing algorithm for a physical cost model need not consider cyclic routes in order to find a shortest anypath route.*

One may also surmise that shortest anypath routes will also be acyclic with a non-physical cost model, since letting a node travel in a loop should only increase the cost of a route. However, this is not the case.

Remark 5.3. In a non-physical cost model, the shortest anypath route can contain cycles.

Example 5.4 (A shortest anypath route that contains a cycle)**.** The network of Figure 5.5 contains three nodes: a source j, a destination 1, and a third node k that may serve as a relay. If we use the (non-physical) delivery probability (E2E) cost model, the shortest anypath route from j to the destination contains a cycle. For clarity we consider directly the end-to-end delivery probabilities, rather than the negative logarithms as per the definition of packet delivery ALC (Definition 4.9). We consider first in Figure 5.5(a) the shortest *acyclic* anypath route. As we computed in Figure 5.1, this route has a packet delivery probability $\frac{9}{16}$. (It is the shortest acyclic anypath route because the only other possibility would be to use only the link from j to 1, giving a lower delivery probability of $\frac{1}{2}$).

Consider now the cyclic route of Figure 5.5(b). To compute the end-to-end delivery probability p_j that a packet originated at node j is delivered to the destination, we must know the delivery probability p_k from node k. However, due to the cycle between j and k, the probability from k depends itself on j. We must therefore compute both probabilities jointly:

$$p_j = p_{j1} + (1 - p_{j1})(1 - p_{jk})\, p_k = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2}\, p_k$$
$$p_k = p_{k1} + (1 - p_{k1})(1 - p_{kj})\, p_j = \frac{1}{4} + \frac{3}{4} \cdot \frac{1}{2}\, p_j$$

Solving these equations gives $p_j = \frac{18}{29}$, giving a higher delivery probability from node j to the destination using this cyclic route than using the acyclic route of Figure 5.5(a). Note that the same argument applies when considering node k as the source: using the cyclic route we obtain $p_k = \frac{14}{29}$, where as the acyclic route going through j would have a lower delivery probability of $\frac{7}{16}$.

Interpretation: routing without buffers

In the previous example, it is surprising that the delivery probability actually *increases* when we allow a packet to go through a cycle. To understand why this makes sense, we must recall again that the model of unreliable links without retransmissions is equivalent to bufferless nodes. Consider the following sequence of events:

1. Node i transmits a packet for the destination node 1.
2. The packet is not received by node 1, however it is received by node k.
3. Node k now transmits the packet to the destination 1.
4. The packet is not received by node 1 , however it is received by node i.
5. The packet has gone through a cycle between nodes i and k.

Note that this cycle may be repeated multiple times, if each transmission by i (respectively k), is not received by the destination but is received by k (respectively i). This is somewhat reminiscent of hot potato routing [35] for wired networks with limited buffer sizes, where it is preferable to forward a packet to a node that is further to the destination than to simply drop it, in the case that the link to the “best” next-hop node is busy.

Note that with ERS-best, the shortest anypath route under the E2E metric does not just contain “some” cycles: nodes are assigned all their neighbors as candidate relays, e.g., $C(i) = N(i)$. Indeed, delivery probability cannot decrease by adding poorly placed nodes to the CRS, because these nodes will only be used if all others have not received a packet. In contrast, the shortest anypath routes under E2E use more restricted candidate relay sets when using the ERS-all policy that we introduce in the next chapter.

Finally, note also that one might add each node i to it’s own candidate relay set, such that a node may prefer to “keep” a packet after a transmission that is not received by suitable nodes. In this case, shortest anypath routes would not have cycles. Of course, in so doing, we would entirely change the model by implicitly assuming nodes with buffers, and we would also have delivery probability equal to 1 for each node that has a path to the destination.

Cost of shortest anypath routes

Our hope is of course that anypath routes *improve* upon the performance of single-path routing, and we shall examine this performance gain in the following chapters. Our performance evaluation is based on simulation and experimentation, and it is not possible to give closed-form *quantitative* results on the performance of anypath routing in the general case. However, it is reassuring to see that *single-path routing can never compute better routes than anypath routing.*

Proposition 5.6. *Call $\mathcal{R}$ the shortest anypath route between two nodes. If there is any node i in $\mathcal{R}$ for which $|C(i)| > 1$, then the shortest anypath route has lower cost than the shortest single-path route. If all nodes i in $\mathcal{R}$ have $|C(i)| = 1$, then the shortest anypath route has equal cost to the shortest single-path route.*

Proof. Consider all possible anypath routes between a source and a destination, and let $\mathcal{R}$ be the one with lowest cost D_R. Consider all possible single-path routes between a source and a destination, and let $\mathcal{S}$ be the one with lowest cost D_S. By definition, a single-path is also an anypath route, and so the set of all possible anypath routes between a source and a destination includes all possible single-path routes. Therefore, if $D_r < D_s$, the shortest anypath route has at least one node with $|C(i)| > 1$. $\square$

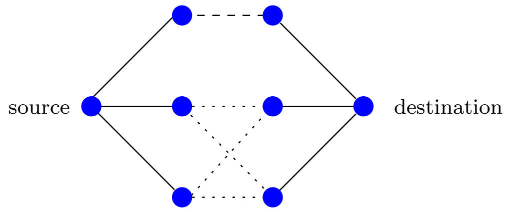
Corollary 5.7. *The shortest anypath distance between two nodes is either smaller than or equal to shortest single-path distance between two nodes.*

5.2.3 Structure of Shortest Anypath Routes

We now state some structural properties of shortest anypath routes. These properties are of interest for their own sake in order to shed light on anypath routing; some shall in addition be useful to prove the correctness of algorithms introduced in the following section.

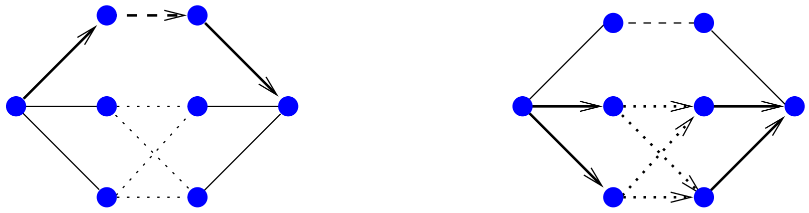
Shortest single-path and anypath routes can be disjoint

We have now seen how anypath routes generalize single-path routes, and that the shortest anypath route is equal to the shortest single-path route when there is no gain to be had from selecting candidate relay sets of size greater than 1. It might appear natural to infer from these facts that the shortest-path route is always included in the shortest anypath route. This is however not always the case, as the following example shows.



(a) Network topology

$p_{ij} = 1$ for solid links, $3/4$ for the dashed link, and $2/3$ for the dotted links.



(b) Shortest single-path route.

(c) Shortest anypath route.

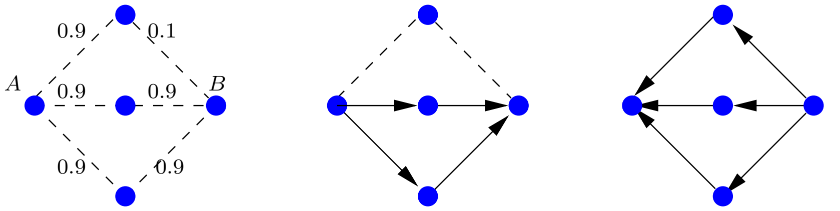
Figure 5.6: Example of a shortest anypath route that does not include the shortest single-path route. The cost metric is expected transmission count (ETX). Illustration of Example 5.5.

Example 5.5 (Shortest single-path route not included in shortest anypath route)**.** Figure 5.6 shows a network with some links (solid) having delivery probability 1, some (dotted) having delivery probability 2/3, and one link (dashed) with delivery probability 3/4. The link metric is expected transmission count (ETX). Solid links therefore have cost $d_{ij} = 1$, dotted links have $d_{ij} = 3/2$, and the dashed link has

$d_{ij} = 4/3$. Since $4/3 < 3/2$, the shortest single-path route will go through the top, for a total route cost of $1 + 4/3 + 1 = 10/3$. With anypath routing, the nodes on the left of the dotted links can use either of the two nodes on the right as their candidate relay set. The anycast link delivery probability is therefore $1 - (1 - 2/3)^2 = 8/9$, and so the expected transmission count of the anycast link is $9/8$, which is lower than $4/3$ for the upper (dashed) link. The shortest anypath route thus goes through the bottom portion of the network and does not include the shortest single-path route. The shortest anypath route has overall cost $1 + 9/8 + 1$, lower than the shortest single path cost.

Asymmetry of shortest anypath routes

With single-path routes, route costs are asymmetric as long as individual links are asymmetric (or equivalently, if the underlying graph is not directed). This property does not hold in general for anypath routes, even when individual links are symmetric. In addition, reversing a shortest anypath route from a source to a destination does not always result in the shortest anypath route from the destination to the source, as the following example shows. While exploring the consequences of this asymmetry is outside the scope of this dissertation, it shall probably impact the design of transport layer protocols that are intended to operate over anypath routes and that require sending packets in both directions between two endpoints.



(a) Network topology (b) Shortest anypath route from A to B. (c) Shortest anypath route from B to A.

Figure 5.7: Example of a shortest anypath route that is not symmetric. Link delivery probabilities are depicted in the left-most figure. The cost metric is expected transmission count (ETX). Illustration of Example 5.6.

Example 5.6 (Asymmetry of shortest anypath routes)**.** Figure 5.7 shows a network with a two end-points A and B, and three intermediate nodes. All links have delivery probability 0.9, except for one link that has delivery probability 0.1. The link metric is expected transmission count (ETX).

Figure 5.7(b) shows the shortest anypath route from A to B. This route does not use the upper node as a candidate relay, because it has a poor connection to B. Given that this upper node has to re-transmit on average 10 times to delivery a packet to B, it is preferable for node A to re-transmit in the rare case that neither of the two bottom relays receives the packet, even if the upper node has received it.

Now let us consider the reverse direction, from B to A. Here, the shortest anypath route uses all three intermediate nodes are candidate relays. Using a smaller candidate relay set would result in a higher ETX to get from B to the candidate relay set, and since all intermediate nodes have the same delivery probability to A, there is no performance hit from using the upper relay (unlike in the opposite direction). Note that removing the upper node from the candidate relay set would result in a route whose cost is only *slightly* higher than the shortest anypath route, since the ETX from B to either of the two lower nodes is already very close to 1 (it is equal to $(1 - 0.1^2)^{-1} = 1.01$). In other words, to send packets from B to A by using the inverse of the shortest anypath from A to B would not result in substantially decreased performance compared to the true shortest anypath from B to A.

In the other direction, sending packets from A to B across the inverse of the shortest anypath from B to A would also not degrade performance substantially, as long as we use ERS-best. This policy would select as effective relay either of the two bottom nodes whenever possible. The only time that we would effectively go through the upper node is when it receives a packet and the lower two do not, which happens with very small probability. In contrast, with ERS-any, we would use the uppermost relay with probability $\frac{1}{3}$, since each of the three relays receives a packet from A with equal probability. The performance gap of using the inverted opposite-direction route would thus be significantly higher with ERS-any than with ERS-all

Sub-shortest anypaths are also shortest anypaths

In single path routing, the sub-paths of a shortest path route are themselves also shortest-paths. For example, if $(n_1, n_2, n_3, \dots n_{k-1}, 1)$ is a shortest single-path route from n_1 to 1, then $(n_2, n_3, \dots n_{k-1}, 1)$ is a shortest-single path route form n_2 to 1, $n_3, \dots n_{k-1}, 1$ is a shortest-single path route form n_3 to 1, and so on. This fact is obvious and is usually not even stated explicitly in the context of single-path routing.

Does a similar property hold for shortest anypath routes? Fortunately, the answer is positive. Before stating this formally, we must extend to anypath routes the notion of subpath used in the previous paragraph. Consider an anypath route $\mathcal{R}$ that traverses a node i (i.e., i is not the source in this route). We say that the *sub-anypath* of $\mathcal{R}$ from node i is the anypath route consisting of node i and all successor nodes of i in $\mathcal{R}$. Sub-anypath routes are illustrated in Figure 5.8.

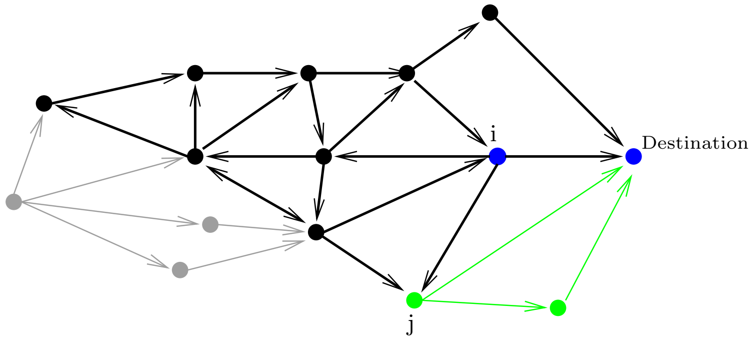


Figure 5.8: Illustration of sub-anypath routes. This shows two sub-routes of the route of Figure 5.3. The first, in green, is the sub-anypath from node j. It is acyclic. The second is the sub-anypath from node i. It contains cycles, and is made up of both the green and the black nodes and links in the diagram.

Having defined a sub-anypath route, we can formally state that the sub-anypath of a shortest anypath route is itself a shortest anypath. In other words, the "obvious" property of single-path routing outlined above carries over to anypath routing.

Proposition 5.8. *Let $\mathcal{R}$ be a shortest anypath route from a source to a destination, and node k be an interior node in $\mathcal{R}$. Call $\mathcal{R}_k$ the sub-anypath route of $\mathcal{R}$ from node k, and define $D_k = Cost(\mathcal{R}_k)$. Then,*

$$D_k = D_k^*,$$

where D_k^* is the shortest anypath distance from node k to the destination.

Proof. Call $\mathcal{T}$ the shortest anypath route from node k to the destination. We have therefore $Cost(\mathcal{T}) = D_k^*$. Since $\mathcal{T}$ is the shortest anypath from k to the destination, we cannot have $D_k^* > D_k$, or otherwise $\mathcal{R}_k$ would be a shorter anypath than $\mathcal{T}$. It now remains to be shown that we cannot have $D_k^* < D_k$. We now proceed by contradiction and assume that $D_k^* < D_k$.

Return now to the shortest anypath route $\mathcal{R}$ that $\mathcal{R}_k$ is a sub-anypath of. If $D_k^* < D_k$, then any packets arriving at k from the source of route $\mathcal{R}$ toward the destination can be forwarded using $\mathcal{T}$. This results in a new route that we call $\mathcal{R}^*$, going between the same source and the destination as route $\mathcal{R}$. To complete the proof we observe that $\mathcal{R}^*$ has lower cost than $\mathcal{R}$, contradicting our initial assumption that $\mathcal{R}$ was a shortest anypath route. ☐

This proposition is a crucial one, and it is fortunate for two reasons that it holds for shortest anypath routes. The first is that it allows to reason recursively about routes, both in order to construct and to prove the correctness of algorithms to find these shortest anypath routes. This is comparable to the case of single-path routing, where constructing a k-hop shortest-path route can be done starting from the shortest $k-1$-hop path, recursively descending so on until $k = 0$. Making use of this property in the anypath context shall be the subject of the next section. The second reason why this proposition is helpful is that without it, each node would potentially be required to compute and keep track of one candidate relay set per (source, destination) pair, rather than having only one set per destination.

Note that this property does not hold, for example, with node-disjoint multipath routing: if all paths of a multipath route between two nodes are disjoint, then there can only be a fork at the source. Consequently, the sub-path of an interior node cannot contain a fork. Is is thus a single-path route, and is different from a multipath route that would be used to reach the destination directly from this node.

Sub-anypaths with non-physical costs and acyclic routes

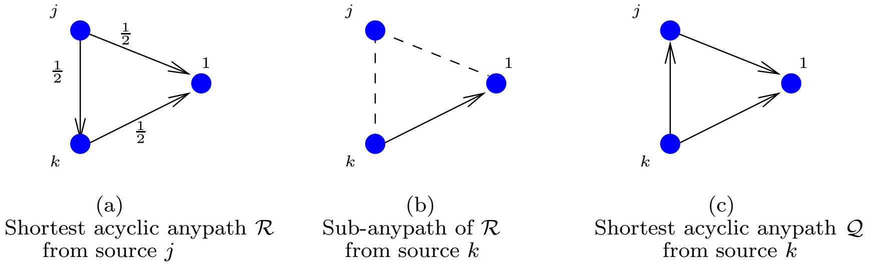
We can now give an additional motivation for why the definition of an anypath route allows for the presence of cycles: this is necessary in order for Proposition 5.8 to hold for physical cost models *as well* as non-physical cost models, thus allowing us to treat both using the same framework and algorithms in the next section. Indeed, if our definition of anypath routes included only *acyclic* directed graphs, Proposition 5.8 would hold true only for physical network models.

Example 5.7 (Proposition 5.8 with non-physical cost model and acyclic routes)**.** We illustrate Proposition 5.8 in Figure 5.9. All links have delivery probability $\frac{1}{2}$. In Figure 5.9(a), the shortest acyclic anypath $\mathcal{R}$ from j to destination 1 is obtained similarly as in Figure 5.5. In Figure 5.9(b), the sub-anypath of $\mathcal{R}$ from source k contains simply the link from k to the destination. In Figure 5.9(c), the shortest acyclic anypath $\mathcal{Q}$ from k to the destination is obtained similarly to $\mathcal{R}$ (or by symmetry). It is not equal to the sub-anypath of $\mathcal{R}$ shown in (b).

5.3 Computing the Cost of Acyclic Anypath Routes

In the previous section, we defined the cost of an anypath route as the expected cost of a trajectory across that route (Definition 5.4).

While the definition of anypath route cost as an expectation is a natural one, and has been sufficient thus far to reason and prove properties of shortest anypath routes, it does not offer much insight into *how* to compute anypath route costs.



(a)
Shortest acyclic anypath $\mathcal{R}$
from source j

(b)
Sub-anypath of $\mathcal{R}$
from source k

(c)
Shortest acyclic anypath $\mathcal{Q}$
from source k

Figure 5.9: With a non-physical cost model, sub-anypaths of a shortest *acyclic* anypath are not always shortest anypaths. The shortest acyclic anypath from node k to the destination is shown in (c); it is not equal to the sub-anypath from node k, shown in (b), of the shortest anypath from node j to the destination. Illustration of Example 5.7.

The direct way to compute the cost of an anypath route $\mathcal{R}$ is to enumerate all possible trajectories across that route, and compute for each trajectory T its cost $c(T|\mathcal{R})$ and probability $\mathrm{P}(T)$. Computing the cost of a trajectory is straightforward using Definition 5.3. However, as mentioned in the previous section, computing the probability of a trajectory is problematic because it depends on the choice that is made by the ERS policy at each anycast forwarding phase. And, at least with the policy ERS-best, the choice itself depends on the relative costs of the sub-anypaths from each node in a CRS.

In this section we propose an alternate method to compute the anypath route cost. This method is valid only for acyclic routes, but it has the advantage of "undoing" the circularity problem arising from the interdependence of $\mathrm{P}(T)$ and interior sub-anypath costs.

The only way to get around this circularity is to start where the ERS-best policy is *not* used: at the node(s) having as candidate relay set the destination itself (and no other nodes). Note that if the route is acyclic, there is at least one such node. We can immediately compute the cost of this node, and then work our way recursively backwards from the destination to the source, computing the costs of interior nodes along the way.

Define T_k to be the set of those nodes in $\mathcal{R}$ from which there is a k-hop path to the destination, and T_0 to be the singleton set consisting of the destination only. A node can be in more than one such set, e.g. in Figure 5.4(d), node c is both in T_1 and T_2, while in Figure 5.4(c) node c is only in T_1.

The algorithm to compute the cost of the anypath route $\mathcal{R}$ is as follows. It proceeds in N iterations, where N is the length (again, in hops) of the longest trajectory (in $\mathcal{R}$) from the source to the destination. The n-th iteration of the recursion consists in computing the distance for all nodes in T_n, taking into account *only* the nodes in $T_{n-1} \cup T_{n-2} \ldots \cup T_0$. Note by $C_n(i)$ the subset of node i's candidate relays for which we have already computed the cost to the destination: $C_n(i) = C(i) \cap (\cup_{i=0}^{n} T_i)$. These nodes can be used in the next step $(n+1)$ to compute the cost at node i. The n-th update step is expressed formally as:

$$\begin{aligned} D_i^n &= d_{iC_{n-1}(i)} + R_{iC_{n-1}(i)}^{n-1}, && \text{if } i \in T_n, \\ D_i^n &= D_i^{n-1} && \text{if } i \notin T_n. \end{aligned} \tag{5.1}$$

where the recursion comes from the fact that $R_{iC_{n-1}(i)}^{n-1}$ is a function of D_k^{n-1}, $k \in C_{n-1}(i)$. The set T_0 contains only the destination itself (node 1), and we set as starting conditions $D_1^0 = 0$. A few comments on this iteration:

- The recursion proceeds backwards, starting at the destination and ending at the source, since the cost at each node depends on the costs of the downstream nodes.
- As a side-effect of computing the cost of an anypath route, we compute the costs from all interior nodes in the route.
- The number of cost assignments performed using (5.1) is at most $\frac{n(n-1)}{2}$, where n is the number of nodes in $\mathcal{R}$.

Example 5.8 (Recursive computation of anypath route cost)**.** We illustrate this recursive computation for an anypath route in Figure 5.10, highlighting at each step in blue the current set of "active" nodes T_n for which we are updating the distances, and in green the set of nodes $(\cup_{i=0}^{n-1} T_i)$ that we can use to compute the values for the "active" nodes.

It now remains to be shown that computing these D_i^n (where the remaining path cost is prominently used) does indeed give us the anypath route cost from Definition 5.4, that made no use whatsoever of remaining path costs.

Proposition 5.9. *Consider an acyclic anypath route $\mathcal{R}$ with a source node i. If N is the length (in hops) of the longest trajectory in $\mathcal{R}$ from i to the destination, then*

$$D_i^N = \sum_{T \in \mathcal{R}} P(T) \cdot c(T|\mathcal{R}) = Cost(\mathcal{R}),$$

where D_i^n are computed as per the iterative step in (5.1)*.*

Proof. We shall prove a more general proposition, namely that for *every* node j in the anypath route, D_j^N is equal to the cost of the sub-anypath of $\mathcal{R}$ with source j.

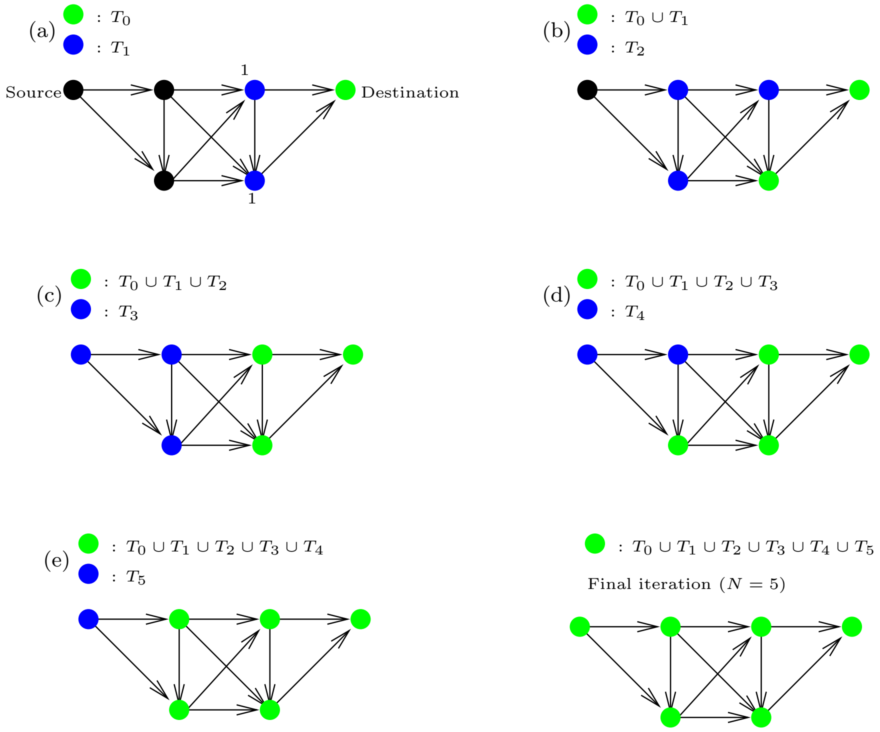

(a)
: T_0
: T_1
Source
1
Destination
1
(b)
: $T_0 \cup T_1$
: T_2
(c)
: $T_0 \cup T_1 \cup T_2$
: T_3
(d)
: $T_0 \cup T_1 \cup T_2 \cup T_3$
: T_4
(e)
: $T_0 \cup T_1 \cup T_2 \cup T_3 \cup T_4$
: T_5
: $T_0 \cup T_1 \cup T_2 \cup T_3 \cup T_4 \cup T_5$
Final iteration ($N = 5$)

Figure 5.10: Recursive computation of anypath route cost. The "active" nodes T_n whose distances are computed in a given iteration are in blue, and the nodes $(\cup_{i=0}^{n-1} T_i)$ whose distances are computed are in green. Illustration of Example 5.8.

Call S_k the set of nodes in $\mathcal{R}$ whose *longest* trajectory to the destination is of length k hops or less. We shall show by induction that for all $j \in \mathcal{R}$

$$\text{if } j \in S_k, \text{ then } D_j^k = Cost(\mathcal{R}_j),$$

where $\mathcal{R}_j$ is the sub-anypath of $\mathcal{R}$ starting from node j. Note that showing this will prove the proposition, because $S_N = \{i\}$.

Case $\mathbf{k = 1}$. Each node j in S_1 has only one possible trajectory to the destination, namely $(j, 1)$. This trajectory is used with probability 1 to go from j to the destination, and has cost equal to d_{j1}. Since each node j in S_1 has $C(j) = \{1\}$, and since the remaining path cost of the destination to itself is 0, Definition 5.4 assigns the cost $Cost(\mathcal{R}_j) = d_{j1}$ correctly to all nodes in S_1.

Induction over k. We now assume that $D_j^k = Cost(\mathcal{R}_j)$ holds for all nodes in

S_i (for all $i \in [1, k-1]$), and must show that it holds equally for nodes in S_k. Consider such a node j belonging to S_k, with candidate relay set $C(j)$. Note that every trajectory in $\mathcal{R}$ from j to the destination must necessarily have as first hop one of j's candidate relays. Note p_h the probability that the first hop of a trajectory from j is node h. The cost of node j is then

$$\begin{aligned}
Cost(\mathcal{R}_j) &\overset{\text{(a)}}{=} \sum_{T \in \mathcal{R}_j} \mathrm{P}(T) \cdot c(T|\mathcal{R}_j) \\
&\overset{\text{(b)}}{=} \sum_{h \in C(j)} p_h \left(d_{jC(j)} + \sum_{T \in \mathcal{R}_h} \mathrm{P}(T) \cdot c(T|\mathcal{R}_h) \right) \\
&\overset{\text{(c)}}{=} d_{jC(j)} + \sum_{h \in C(j)} p_h \sum_{T \in \mathcal{R}_h} \mathrm{P}(T) \cdot c(T|\mathcal{R}_h) \\
&\overset{\text{(d)}}{=} d_{jC(j)} + \sum_{h \in C(j)} p_h \, Cost(\mathcal{R}_h) \\
&\overset{\text{(e)}}{=} d_{jC(j)} + \sum_{h \in C(j)} p_h D_h^k \\
&\overset{\text{(f)}}{=} d_{jC(j)} + R_{jC(j)} = D_j^k,
\end{aligned}$$

where (a) and (d) are from Definition 5.4, (b) is from the fact that every trajectory in $\mathcal{R}$ from j to the destination must necessarily have as first hop a node in $C(j)$, (c) is because $\sum_{h \in C(i)} p_h = 1$, and (e) follows jointly from our induction hypothesis and our starting assumption that the route is acyclic, giving $C(j) \subseteq S_k$.

The final step (f) to complete our induction is to note that p_h is implicitly conditional on the event "at least one node in $C(j)$ receives the packet", since we are considering a trajectory that arrives at the destination. With this in mind, we can see that $\sum_{h \in C(j)} p_h D_h^k$ is equivalent to our definition of remaining path cost and conclude that $D_j^k = Cost(\mathcal{R}_j)$. $\square$

5.4 Finding Shortest Anypath Routes

We now show an algorithm to compute shortest anypath routes. We assume throughout this section the use of a *physical* cost model; therefore we have from Proposition 5.4 that shortest anypath routes in this section are acyclic.

The algorithm is based on the classical Bellman-Ford algorithm, and its development resembles the one for the single-path case. It is somewhat remarkable that the upper-bound on the algorithm's convergence time (in number of iterations) is the same for anypath routing as for single-path routing. Not surprisingly the complexity of the anypath algorithm is however greater, because it must compute at each iteration a set $C(i) \subseteq N(i)$ for each node and not just a single next-hop.

How does a node select which of its neighbors should be candidate relay nodes? With anypath routing, the expression to minimize becomes the sum of the anypath link cost and the remaining path cost $D_i = d_{iC(i)} + R_{iC(i)}$, and this sum must be minimized over all possible subsets $J \subseteq N(i)$:

$$D_i = \min_{J \in 2^{N(i)}} [d_{iJ} + R_{iJ}]. \tag{5.2}$$

This is called the *Bellman's Equation* for anypath routing, and it represents the steady-state of the anypath Bellman-Ford algorithm that we now describe.

5.4.1 Bellman-Ford Anypath Routing with Physical Costs

Consider the problem of finding the shortest anypath from every node to the destination (node 1) under a physical cost model. Recall our convention that $d_{ij} = \infty$ if (i, j) is not an link of the graph, and $d_{ii} = 0$. We shall compute the shortest anypath routes (and their distances) iteratively as follows. At each iteration, we update the value D_i^h at each node i, where h is the iteration index. This D_i^h is the shortest-anypath distance estimate from i to the destination at the h-th iteration, and we shall show that the sequence converges toward the shortest-anypath distance D_i. By convention, we take

$$D_1^h = 0, \qquad \text{for all } h. \tag{5.3}$$

One iteration step consists of updating the estimated distance from each node:

$$D_i^{h+1} = \min_{J \in 2^{N(i)}} [d_{iJ} + R_{iJ}^h] \qquad \text{for all } i \neq 1, \tag{5.4}$$

where R_{iJ}^h is the remaining path cost computed using the distances D_j^h, $j \in J$ from the previous iteration. Our definition of this algorithm is completed by noting the initial conditions:

$$D_i^0 = \infty, \qquad \text{for all } i \neq 1.$$

To show that this algorithm computes the shortest anypath distances, we must show that it terminates. Termination occurs when

$$D_i^h = D_{i-1}^h, \qquad \text{for all } i.$$

In the following, a $(\leq h)$ anypath route is a route whose longest trajectory contains at most h hops. A shortest $(\leq h)$ anypath route from a node i is a shortest anypath route from i to the destination, subject to the constraint that the longest trajectory in the anypath route traverses at most h hops.

Theorem 5.10. *The anypath Bellman-Ford algorithm* (5.4) *computes, at iteration h, the shortest ($\leq h$) anypath distances from each node to the destination. Furthermore, the algorithm terminates after at most $h^* \leq |\mathcal{N}|$ iterations, and at termination, $D_i^{h^*}$ is the shortest-anypath distance from i to the destination.*

Proof. We prove the first part of the proposition by induction over h.
Case $\mathbf{h = 1}$. Using (5.4) and our initial conditions, we have

$$D_i^1 = d_{i1}, \qquad \text{for all } i \neq 1,$$

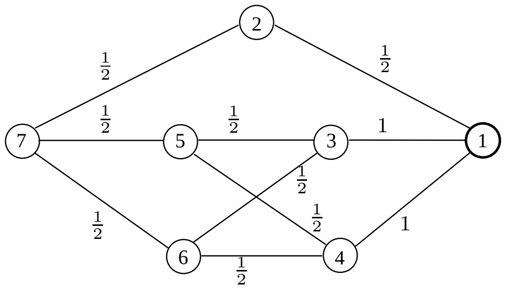
which is indeed the shortest (≤ 1) anypath distance to the destination.
Induction over h. We assume that D_i^h is equal to the shortest ($\leq h$) anypath distance from i to 1, and must show that D_i^{h+1} is equal to the shortest ($\leq h+1$) anypath distance. There are two possible cases for each node i. The first is that the shortest ($\leq h+1$) anypath route from i to 1 contains a longest trajectory with h or less hops. We call this route $\mathcal{R}_i^h$, and in this case we have $Cost(\mathcal{R}_i^h) = D_i^h$. The second possible case is that the shortest ($\leq h+1$) anypath route from i to 1 contains a longest trajectory with $h+1$ hops. Call this route $\mathcal{R}_i^{h+1}$. It has cost

$$Cost(\mathcal{R}_i^{h+1}) = d_{iC(i)} + R_{iC(i)}$$

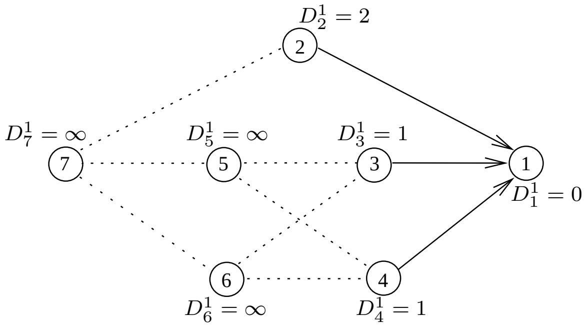
This route consists of $|C(i)|$ links from i to each node in its CRS $C(i)$, and then of $|C(i)|$ sub-anypath routes from each node in $C(i)$ to 1 that each have a h-hop longest trajectory. From Proposition 5.8, we know that these sub-anypath routes must be shortest anypath routes. Given this structure, there is no possible candidate relay set among i's neighbors that has a lower cost to reach the destination with ($\leq h$) trajectories:

$$\begin{aligned} Cost(\mathcal{R}_i^{h+1}) &= d_{iC(i)} + R_{iC(i)} \\ &= \min_{J \in 2^{N(i)}} [d_{iJ} + R_{iJ}^h] \\ &= D_i^{h+1}. \end{aligned}$$

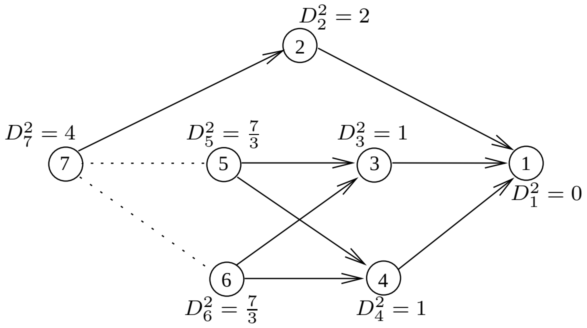
Calling S_i^{h+1} the shortest ($\leq h+1$) anypath route length from i to 1, these two cases



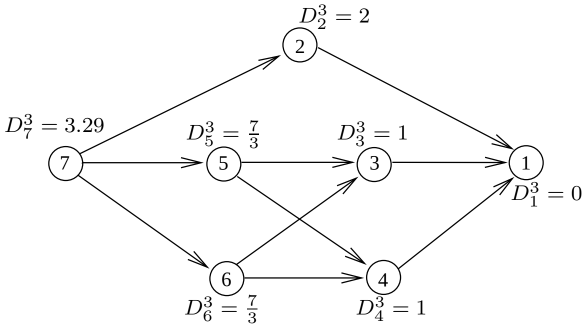
(a) Initial graph with link probabilities annotated. The destination is node 1.



(b) $h = 1$: shortest (≤ 1) anypath routes.



(c) $h = 2$: shortest (≤ 2) anypath routes.



(d) $h = 3$: shortest (≤ 3) anypath routes.

Figure 5.11: Illustration of the Bellman-Ford algorithm computing shortest anypath routes from every node to the destination.

thus give:

$$\begin{aligned} S_i^{h+1} &= \min\left\{Cost(\mathcal{R}_i^h), Cost(\mathcal{R}_i^{h+1})\right\} \\ &= \min\left\{D_i^h, \min_{J\in 2^{N(i)}} [d_{iJ} + R_{iJ}^h]\right\} \\ &= \min\left\{D_i^h, D_i^{h+1}\right\} \\ &= D_i^{h+1}, \end{aligned}$$

and so D_i^{h+1} is the shortest anypath distance from i to 1.

The second part of the proposition follows simply from the first part and the fact that in a network with $|\mathcal{N}|$ nodes, the longest possible path has at most $|\mathcal{N}| - 1$ hops.

□

The development above focused only on computing the shortest anypath distances to the destination. The candidate relay sets are found as part of the iteration step:

$$C(i) = \arg \min_{J\in 2^{N(i)}} d_{iJ} + R_{iJ}.$$

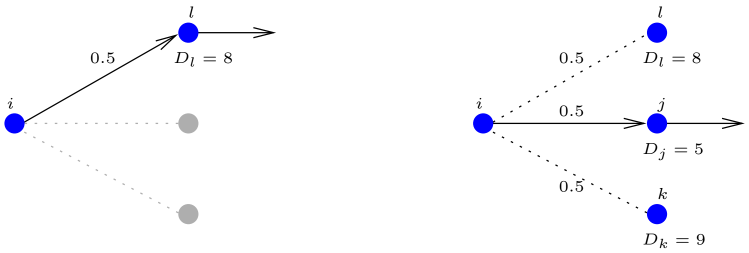
We finally note that the anypath Bellman-Ford algorithm generalizes single-path Bellman-Ford, and similarly can be implemented in a distributed setting, with nodes asynchronously recomputing their shortest-anypath distance and advertising it to their neighbors. Remark also that we can extend the Dijkstra algorithm to compute anypath routes in a similar manner as we have done with the Bellman-Ford.

Remark 5.9. With an anycast link cost that is *unicast-equivalent* (Property 4.5), the anypath Bellman equation (5.2) is equivalent to the single-path Bellman equation (3.3).

The best CRS is not just "closer nodes"

At a first look at the anypath Bellman-Ford algorithm, one might be tempted to infer that at each iteration, node i should simply take as candidate relay set those nodes which have a lower distance to the destination than i, namely the set of nodes k for which $D_k^h < D_i^h$. Under a physical cost model, choosing this set as the CRS does indeed always result in a lower cost D_i^h, by the very definition of a physical cost model. However, this set is *not* in the general case equal to the set that minimizes (5.2).

Example 5.10 (Bellman-Ford iteration does not always select all neighbors with lower cost)**.** We illustrate this with the example of Figure 5.12. All links have delivery



(a) $C(i) = \{l\}$
$D_i = d_{il} + D_l = 10$

(b) $C(i) = \{j\}$
$D_i = d_{i\{j\}} + R_{i\{j\}} = 7$

Figure 5.12: The best CRS does not simply contain all nodes that are closer to the destination. Illustration of Example 5.10.

probability $\frac{1}{2}$ and nodes use the ETX metric. Initially, only node l has advertised its route, and so node i has as CRS $C(i) = \{l\}$, with current cost estimate $D_i^h = 10$. In Figure 5.12(b), nodes j and k come online and advertise their distances. With link delivery probabilities equal to $\frac{1}{2}$, the anycast link cost to the set containing both nodes is $d_{i\{j,k\}} = \frac{4}{3}$. The remaining path costs when using either of the singleton sets are trivially $R_{i\{j\}} = D_j = 5$ and $R_{i\{k\}} = D_k = 9$, and using the equation for the remaining path cost (4.11), we can compute that the remaining path cost for the set containing both nodes is $R_{i\{j,k\}} = 6.33$. We therefore have:

$$\begin{aligned} d_{i\{j\}} + R_{i\{j\}} &= 7 \\ d_{i\{k\}} + R_{i\{k\}} &= 11 \\ d_{i\{j,k\}} + R_{i\{j,k\}} &= 7.66, \end{aligned}$$

and so the set chosen by iteration $h+1$ of the Bellman-Ford algorithm is $\{j\}$ and is different than the set of nodes $\{j,k\}$ satisfying $D_x^h < D_i^h$.

Note that while we have $D_k^h = 9 < 10 = D_i^h$, the sum $d_{i\{k\}} + R_{i\{k\}} = 11$ is not lower than D_i^h, and so one might deduce that this is a required feature of our example. However, a similar example can be constructed with all nodes having $d_{i\{k\}} + D_k^h < D_i^h$. For example, take $D_j^h = 3$ and $D_j^h = 7$, giving $R_{i\{j,k\}} = 6.66$.

On the tension between decreasing ALC and increasing RPC

It is worth pointing out that when computing the $C(i)$ that minimizes the Bellman equation (5.2), there is an inherent tension between decreasing the anycast link cost and increasing the remaining path cost.

Assuming that we have an *always-decreasing* anycast link cost, the tension is the following. On the one hand, we would like to take a CRS that is as large as possible, so as to reduce the first component d_{iJ} that goes into the overall cost D_i. In fact we minimize the ALC by taking the entire neighborhood as our candidate relay set: $C(i) = N(i)$.

On the other hand, minimizing the remaining path cost R_{iJ} pushes us in exactly the opposite direction: we would like the candidate relay set to contain only nodes with low costs D_j to the destination, and so we would like to make it as small as possible. In the extreme, if the nodes $j \in N(i)$ are labelled by increasing distances D_j (i.e., $D_1 < D_2 < D_3 < \ldots < D_{|N(i)|}$), then we minimize R_{iJ} by taking $J = \{1\}$. (Of course, for any n nodes having the same cost D_j, then adding all n nodes to the candidate relay set does not decrease the remaining path cost).

This tradeoff is at the heart of anypath routing. Where the optimal tradeoff point lies depends on the anycast link cost model that we are considering. In a model where the anycast link cost decreases only slightly when we add a node, the candidate relay sets will on average contain fewer nodes than in a model where the anycast link cost decreases faster with CRS size.

Note that *exactly the same tradeoff exists in the single-path case* to minimize the Bellmann equation (3.2). Here, minimizing the next-hop cost d_{ij} requires selecting the "closest" node among our neighbors, and this node is in the general case not the same as the one minimizing the next-hop destination distance D_j. The key difference is in the size of the solution space: in the single-path case we have only $|N(i)|$ next-hop choices to compare, whereas in the anypath case we have up to $2^{|N(i)|}$ possible candidate relay sets. In the next section, we take a closer look at this solution space and ways to reduce the computational overhead required to search through it.

5.4.2 Reducing the Search Space

The search space for anypath routing is significantly larger than for shortest path routing: for a node i with $n = |N(i)|$ neighbors, there are $2^n - 1$ possible non-empty subsets from which to choose the candidate relay set $C(i)$. This means that the effort to compute the Bellman iteration at each node grows exponentially in the neighborhood size. For networks with reasonably small densities (e.g., $|N(i)| < 10$) this computation is feasible. However, for extremely dense networks, the time and complexity required to exhaustively explore the entire solution space of candidate sets will become overwhelming.

Distance filtering

A first reduction comes under the assumption of a physical cost model. With such a model, we can do distance filtering and reduce the size of the set $N(i)$. Define the set of nodes $\check{N}(i) \subseteq N(i)$ consisting of all the neighbors k of i for which $D_k < D_i$. Then, we have that

$$\min_{J \in 2^{\check{N}(i)}} d_{iJ} + R_{iJ} = \min_{J \in 2^{N(i)}} [d_{iJ} + R_{iJ}],$$

where the proof follows immediately from the definition of the physical cost model that says that adding any node in $N(i) \setminus \check{N}(i)$ to the candidate relay set can only increase the distance D_i. So, we can take only the neighbors with lower distance to the destination than ourselves $D_k < D_i$ and perform the Bellman minimization over the nodes in this set. Of course, the reduction in size of the filtered set $|\check{N}(i)|$ relative to the neighborhood size $|N(i)|$ will be different at each node and in each network. Nonetheless, intuitively we expect that for most nodes, approximately half of their neighbors are closer to the destination and half are further, and so a rough approximation is $|\check{N}(i)| \simeq \frac{1}{2}|N(i)|$. This simplification is therefore helpful in practice (e.g., taking us from a search space of size 1024 to 32 when $|N(i)| = 10$), but it does not modify the exponential scaling behavior of the solution space size.

Subset ordering

While in the general case we must resort to heuristics to reduce the exponential scaling of the solution space, we are fortunate that the Bellman equation simplifies considerably, under a set of conditions that hold for some cases of practical interest. These conditions essentially require that there be an ordering on all the 2^n possible subsets of $N(i)$, that this ordering depend only on the D_i, and that the ordering be the same as the ordering of candidate relay sets that results from the costs $d_{iJ} + R_{iJ}$ of (5.2).

The first condition ($\mathbf{H_1}$) states that the remaining path cost decreases when we *add* a node to the candidate relay set with distance to the destination inferior than that of all nodes already in the set. The second condition ($\mathbf{H_2}$) states that the anycast link cost depends only on the size of the candidate relay set, and that this cost decreases as the set grows. Note that $\mathbf{H_2}$ is a stronger condition than the *always-decreasing* property (4.3). The third condition ($\mathbf{H_3}$) states that the remaining path cost decreases when we *replace* a node in the CRS with another node having lower distance to the destination. These conditions are formally defined hereafter:

$\mathbf{H_1}$: For all $k \notin J$ and $J' = J \cup \{k\}$,

$$R_{iJ'} < R_{iJ} \text{ if } D_k < D_j \text{ for all } j \in J.$$

$\mathbf{H_2}$: For all J and J',

$$\begin{aligned} d_{iJ'} &< d_{iJ} \text{ iff } |J'| > |J|, \text{ and} \\ d_{iJ'} &= d_{iJ} \text{ iff } |J'| = |J|. \end{aligned}$$

$\mathbf{H_3}$: For all $l, k \in J$ s.t. $D_l < D_k$,

$$R_{iJ'} < R_{iJ}, \text{ where } J' = J \setminus k \cup \{l\}.$$

We summarize the validity of these three conditions for different network models in Table 5.1. Condition $\mathbf{H_1}$ holds for all models: by adding a node with distance inferior to those already in the set, we can only reduce the remaining path cost. Condition $\mathbf{H_2}$ holds for any network where the packet reception probabilities on all links are uniform across all links: in that case the cost to reach a candidate set is determined by the size of that set. If the packet reception probabilities are not equal, then a larger CRS may have higher anycast link cost than a smaller one, if this larger CRS contains links with lower reception probabilities. Finally, condition $\mathbf{H_3}$ holds for the same models as $\mathbf{H_2}$.

Link model	$\mathbf{H_1}$	$\mathbf{H_2}$	$\mathbf{H_3}$
Always-on unreliable links	Yes	No	No
Always-on unreliable links with ARQ	Yes	No	No
Asynchronous duty cycled links with ARQ	Yes	Yes	Yes
Synchronous duty cycled links	Yes	Yes	Yes
Always-on unreliable links, equal p_{ij}'s ($p_{ij} = p$)	Yes	Yes	Yes
Always-on unreliable links with ARQ, equal p_{ij}'s ($p_{ij} = p$)	Yes	Yes	Yes

Table 5.1: Validity of conditions $\mathbf{H_1}, \mathbf{H_2}$, and $\mathbf{H_3}$ for different network models.

Assume now w.l.o.g. that the n nodes in $N(i)$ are sorted by their distance to the destination, i.e., that $D_1 < D_2 < \ldots < D_n$.

Proposition 5.11. *In a network model satisfying* $\mathbf{H_1}, \mathbf{H_2},$ *and* $\mathbf{H_3}$*, the candidate relay set* $C(i)$ *minimizing the Bellman equation* (5.2) *is of the form* $\{1, 2, \ldots, j\}$*, where* $1 \leq j \leq |N(i)|$*.*

Proof. For conciseness we denote by J the set $C(i)$ that minimizes (5.2).

Note that the proposition is equivalent to saying that (a) J is a sequence of nodes $(k, k+1, k+1, \ldots l)$, and (b) J includes node 1. We then proceed by contradiction to show that both of these properties hold under the hypotheses $\mathbf{H_1}, \mathbf{H_2},$ and $\mathbf{H_3}$.

To show that property (a) must hold, consider a set J minimizing (5.2) that is of the form $(\ldots, k, k+l, \ldots)$, with $l > 1$. This sequence does not satisfy property (a) since it has a gap between k and $k+l$. Consider now the set J' where node $k+l$ has been replaced by node $k+l-1$ in J: $J' = J \setminus \{k+l\} \cup \{k+l-1\}$. By $\mathbf{H_2}$, we have that $d_{iJ'} = d_{iJ}$, and by $\mathbf{H_3}$, we have that $R_{iJ'} < R_{iJ}$. Therefore $d_{iJ'} + R_{iJ'} < d_{iJ} + R_{iJ}$, contradicting our initial hypothesis that J minimizes the Bellman equation. The set minimizing the Bellman equation must therefore satisfy property (a).

To show that property (b) must hold, consider a second set J whose smallest element is k ($k > 1$). This set therefore does not satisfy property (b). Assume that J minimizes the Bellman equation. Now, consider the set $J' = J \cup \{1\}$. By $\mathbf{H_2}$ we have $d_{iJ'} < d_{iJ}$, and by $\mathbf{H_1}$, we have $R_{iJ'} < R_{iJ}$, and therefore $d_{iJ'} + R_{iJ'} < d_{iJ} + R_{iJ}$. The set minimizing the Bellman equation must therefore also satisfy property (b), and so it is of the form $\{1, 2, \ldots, j\}$.

□

Corollary 5.12. *In a network model satisfying* $\mathbf{H_1}, \mathbf{H_2},$ *and* $\mathbf{H_3}$*, finding the* $C(i)$ *that minimizes the Bellman equation* (5.2) *requires only searching through* n *possible sets.*

This reduction of search space size from 2^n to n does require to first sort the nodes of $N(i)$ in increasing order of D_i. But since the complexity of this sort increases as $n \log(n)$, the gain remains significant. Since duty-cycled links are primarily intended to model networks made of low-power wireless sensor devices, and since these devices have very low computational capacity, it is fortunate that such a simplification is possible in this scenario.

5.5 Summary

In Chapter 2, we described how existing anypath routing approaches are inadequate. This chapter has presented our proposed solution, which we believe improves on prior work in the following ways. First, it does not use geographic positions, and thus avoids the pitfalls described in Chapters 1 and 2. Second, it provably finds the anypath routes with lowest cost, and thus performs better than methods based on single-path or multi-path routing. Third, it is based on Bellman-Ford, and can be implemented in a distributed setting, with nodes asynchronously recomputing their shortest-anypath distance (using eq. (5.4)) and advertising it to their neighbors.

Chapter 6

Performance and Variations of Anypath Routing

In the previous chapter we introduced anypath routing and showed how it is possible to compute the shortest anypath routes in a distributed manner. We were also able to prove that a shortest anypath route always has lower cost than a shortest single-path route[1].

Of course, while it is good to know that anypath routing finds lower cost routes than single-path routing, this property is lacking a quantitative characterization. It is necessary to provide a more complete performance evaluation, firstly to assess by what margin we can expect anypath routes to be cheaper than single-path ones, and secondly in order to explore and understand tradeoffs that will guide the design of real anypath routing protocols.

With this in mind, our goal in this chapter is to:

- Evaluate the performance of the shortest anypath routing algorithm of Chapter 5,
- explore variants of this algorithm that compute "suboptimal"[2] anypaths, and evaluate how good (or bad) the anypath routes that they compute are,
- motivate how and why these derived algorithms may allow significant reductions in the cost of the relay arbitration and relay notification mechanisms, and

---

[1] Unless of course the shortest anypath route is itself single-path.

[2] We use here the term "optimal" anypath routing to refer to the algorithm of Chapter 5, that is optimal in the sense of computing the shortest anypath routes.

- evaluate the robustness and stability of anypath routing in the face of fluctuations and imperfect protocol information.

Summary of contributions

Besides the simulation-based performance study of anypath routing, the contributions of this chapter include:

- Introducing two new effective relay selection (ERS) policies for selecting the effective relay, called ERS-any and ERS-all, and showing that these policies reduce protocol overhead at the cost of increased-length anypath routes,
- showing that in certain cases, the increase in route length is negligible, allowing us to use either of these policies and have a significant simplification and reduction in overhead of a protocol implementation.

6.1 Evaluating Anypath Routing

In this section we describe in more detail our evaluation methodology. We first discuss the protocol mechanisms that are required to implement anypath routing in a practical system, and then describe and motivate our approach to evaluate the performance of anypath routing algorithms.

6.1.1 On Policy vs Mechanism

The core function of an anypath routing protocol is to compute the candidate relay set $C(i)$ at each node i. It is important to emphasize that the choice of CRS $C(i)$ is a *policy*, as is the effective relay selection (e.g., ERS-best). These policies are carried out in practice using protocol *mechanisms*, and these mechanisms are (conceptually at least) separate from the policies that they carry out. We now define three key protocol mechanisms that are required to implement anypath routing.

Learning link costs and neighbor distances

The first mechanism is the one that gathers information necessary to compute anypath routes; that is, the information driving the selection of candidate relay sets at each node using the Bellman-Ford algorithm (5.4). At least two kinds of information must be learned by a node i to compute its candidate relay set: the packet delivery probability p_{ik} to each neighbor k, and the estimated distance D_k from each neighbor to the destination. Different techniques exist to estimate the former; a simple example is by means of periodic local broadcasts containing an increasing sequence number allowing a receiver to compute the number of missed packets in a stream. The distances D_k can be learned also by means of periodic message exchanges, whereby nodes update their neighbors with their current estimated distance to the destination. In addition, a node must learn the distances d_{ij} to each neighbor. In most cases, these are either fixed (asynchronous duty cycling) or can be computed from p_{ij} (expected transmission count), but in general they can require additional communication, for example to learn schedules with synchronous duty cycling.

Note that such a mechanism for learning link costs and neighbor distances is also required by single-path routing and is therefore not specific to anypath; for this reason we do not discuss it in detail in this chapter. However, we can already comment that the cost of periodic message exchanges is proportional to their frequency. A fair comparison between anypath and single-path routing must therefore take into account the frequency at which update messages must be exchanged in either algorithm. We shall return to this point in Section 6.6 of this chapter when we study the stability of anypath routing metrics.

Relay notification

A second protocol mechanism is required to communicate to those nodes in $C(i)$ that they are candidate relays for packets from i, or equivalently, to inform the neighbors *not* in $C(i)$ that they are not candidate relays.

Definition 6.1. Relay Notification *(RN) is the mechanism used by each node* i *to inform nodes in* $C(i)$ *that they are candidate relays for packets forwarded by* i *toward a given destination.*

Similar to the next-hop field in a unicast packet, a simple RN mechanism is for example to add to packets forwarded by i a header listing the nodes of $C(i)$. More sophisticated mechanisms are possible and will be discussed in Chapter 7.

Relay arbitration

Just as the candidate relay set of a node must be communicated to that node's neighbors using some protocol mechanism, the effective relay selection policy also requires some protocol machinery. Since a candidate relay node receiving a packet does not know which other nodes may have received it, this node can thus not decide on its own whether or not to become the effective relay (except, of course, in the trivial case that it is the only node in the candidate relay set).

Definition 6.2. Relay Arbitration *(RA) is the distributed protocol that executes the ERS policy. It informs the chosen node(s) in the available relay set* $A(i)$ *that it should relay the packet, and the other nodes that they should not. Relay arbitration is only needed in conjunction with receiver-driven anycast forwarding, and not with sender-driven anycast forwarding.*

A simple though inefficient example of relay arbitration is a centralized scheme where all nodes in $A(i)$ send an acknowledgement (randomized over some interval to reduce collisions) back to i, and i then informs the selected node that it is chosen. Again, more sophisticated mechanisms are possible and will be discussed in Chapter 7.

6.1.2 Methodology

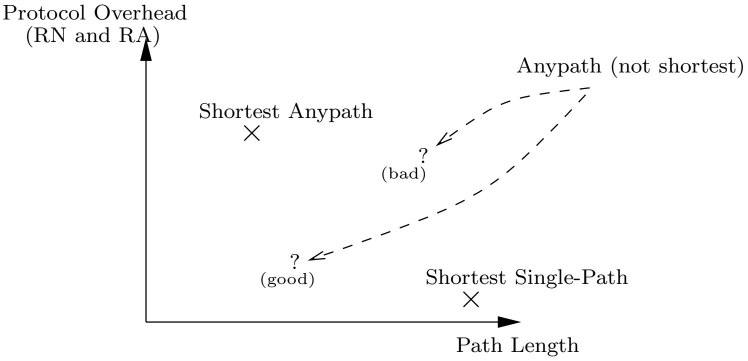


Figure 6.1: Tradeoff between path length and protocol overhead. Shortest anypath routing finds less costly paths than single-path but has higher protocol overhead, in particular with receiver-driven anycast forwarding and wireless links. Can we find suboptimal anypath routing algorithms that trade off a small increase in route length for a large reduction in protocol overhead?

As we have outlined above, the protocol mechanisms implementing relay notification and relay arbitration will unavoidably come with some overhead, in the form of increased latency, increased energy use, or reduced throughput. Furthermore, this overhead is not present in the case of single-path routing, since these two mechanisms are not necessary (or only in a trivial form, i.e., relay notification is done by placing a next-hop address in a packet header).

At the end of the day (and at the end of this dissertation), the key will therefore be to answer the question: *is the protocol overhead of anycast forwarding smaller than the forwarding gains arising from the use of shortest anypath routes?* Unfortunately, answering this question directly requires making specific assumptions on the design of the RN and RA protocols. Many different schemes for these protocols are possible, and the cost of each one is closely tied to the characteristics of the physical layer employed. At this point, two design and evaluation methodologies are possible to answer the above question.

Separate optimization

The first possible methodology is to transcribe the anypath algorithm of Chapter 5 directly into a protocol, and then attack the problem of designing practical mechanisms to perform RA and RN that have a cost lower than the gain of anypath routes. In the best case, the result will be a working protocol that out-performs single-path routing. This would be a notable achievement in itself, but unfortunately it would probably not provide a solution that is widely applicable, given that any implementation of RA or RN will be highly link-specific, due to the large differences between wireless link technologies illustrated in Table 2.1.

Of course, it may turn out that we cannot implement RA and RN mechanisms with a cost that is sufficiently low for anypath routing to be worthwhile. Or, it may turn out that the high cost of RA and RN require us to impose changes upon the application and/or transport layers in order to see an improvement due to anypath routing. An example of this latter case is the XOR protocol, where packets are batch-sent in groups of 100 in order to spread the cost of RA and RN over a large number of packets. Unfortunately this batched send approach comes at the cost of increased latency. This latency may remain acceptable if packet injection rates are high enough, but in the case of low-rate, low-power networks, waiting for a node to have a large number of packets before sending will increase latency significantly, and furthermore buffering resources are highly limited in the target platforms we are considering.

The problem of this methodology is that it implicitly assumes that the design of an anypath routing protocol is *separable*: that we should first design the optimal algorithm to compute the shortest-possible anypath routes, and then separately design protocol mechanisms with minimal overhead to carry out this policy. The use of this “divide-and-conquer” approach is legion in many engineering disciplines, and so it would be a reasonable choice to follow it and assume that it will lead to the best possible protocol design. However, by immediately adopting the shortest anypath routing algorithm of Chapter 5, we run the risk of painting ourselves into a corner and neglecting alternatives that may be more interesting.

In other words, there may be variants on shortest anypath routing that compute slightly suboptimal routes, but that allow sizeable reductions in the overhead of anycast forwarding mechanisms, such that overall performance is improved.

Joint optimization

For this reason, we eschew the approach of separate optimization, and adopt the following alternative methodology: we study variants of anypath routing that are suboptimal (in the sense of not always computing the *shortest* anypath routes) but that may allow implementing significantly cheaper RA and RN mechanisms than with optimal anypath routing, or even in some cases do away with these mechanisms altogether. Of course, we do not expect that the reduction route quality will always be compensated by the reduction in protocol overhead. However, the key is to expose explicitly the different performance tradeoffs, in a way that offers general guidelines to protocol designers seeking to pick and match RA and RN protocol variants with anypath routing algorithms.

While we shall quantify the difference in route costs arising from the use of suboptimal routing variants, we do *not* in this chapter seek to quantify the RN and RA costs; we do however give a qualitative argument as to why they might be lower with certain anypath variants than others. The reason to avoid trying to quantify RN and RA costs at this point is that, as we have remarked above, these costs are highly implementation- and platform-specific. By exposing the costs of different anypath algorithms *in isolation* from the costs of RN and RA, we provide a characterization that can be valuable for the implementation of different anypath protocols for different platforms, where the cost of specific RN and RA implementations will help guide the choice of shortest anypath routing or one of its variants exposed in this chapter. In essence, we seek to lay down a design framework and expose cost tradeoffs which can then be used to design real protocols for specific applications and devices.

We can still note for now that whatever protocol design is used for RN and RA, some broad tradeoffs do apply. The first is that the overhead of both mechanisms will increase with the size of the candidate relay set. At one extreme, single-path routing has a singleton CRS, only needs to list *one* next-hop node in its packet header, and does not need relay arbitration. At the other extreme, with a very large CRS, the list of candidate relays becomes more costly to transmit, and relay arbitration is required increasingly often, and with an increasingly large set of nodes to arbitrate between.

6.1.3 Performance Evaluation

The experimental results of this chapter are obtained through simulation. These simulations are intentionally carried out at a high level of abstraction and idealization; their aim is to highlight properties of anypath routing and allow us to explore high-level tradeoffs. Of course, the best way to evaluate the effect of realistic wireless channels on our scheme is to deploy it on a live network (at the cost of a loss of generality); the lessons learned here shall be used to guide the protocol design and evaluation of Chapter 7.

Our simulation set up is as follows. We first fix the number of nodes n, the dimension d of the Euclidean space that the nodes inhabit, and the average degree ρ. This graph density is the average number of neighbors per node. Nodes are distributed at random with a uniform distribution over a hypercube (i.e., a line segment for $d = 1$, a square for $d = 2$, and a cube for $d = 3$) in the d-dimensional space, and any two nodes whose Euclidean distance is less than 1 are neighbors. The side length l of this hypercube in which nodes are placed is computed as follows in order to obtain the density ρ:

$$l = \left(\frac{n \cdot v(d)}{\rho} \right)^{\frac{1}{d}},$$

where $v(d)$ is the volume of a unit hypersphere in d dimensions. All simulations described in this chapter have $d = 2$ except when noted otherwise.

This model of connectivity, where nodes are neighbors if and only if the Euclidean distance between them is lower than 1, is known as the *unit-disk* graph [16]. The unit disk graph is a strong idealization of wireless connectivity; due to its simplicity it has been used extensively as a model of wireless networks both for simulation [12, 36, 62] and analytical purposes [22,58,117]. In the context of a simulation study, the unit disk graph model is useful not only because it has more modest computational requirements than others, but because it is a parameter-free model, unlike more complex models which can have several parameters, each representing a “knob” that can be tweaked independently. Here, the setup is simple enough that all experiments are easily reproduced, and we are not required to select arbitrary values for simulation parameters.

Our choice of the unit disk graph for this simulation section is not intended to convey that this model is a faithful representation of realistic radio connectivity effects. In fact, it has been observed empirically [32] that connectivity in real multi-hop wireless networks is less regular than a unit disk graph: while connectivity does inevitably decay at medium and large distances, at local scales the role of other factors such as propagation, fading, or mobility is equally if not more important than distance. It should be said however that the idealized nature of the unit disk graph becomes a limitation in particular when it is used to study problems where the physical layout of nodes is highly important (e.g., geographic routing [51]), or where interference between concurrent transmissions (e.g., congestion and collisions) [8, 49] is a central performance issue. Neither of these attributes apply to this work: anypath routing works over the logical graph topology rather than geographic layout, and we are not considering saturating traffic models.

The units of route lengths are implicit from the link model: for the transmission-count metric the unit is packet transmissions, for asynchronous duty cycling it is transmission *duration*, counting both preambles and packets, and for synchronous duty cycling it is latency.

Confidence intervals

We performed multiple independent replications of each experiment and compute the mean, variance, and confidence intervals. From a series of measurements $X_1, X_2, \ldots X_n$, we compute the mean and standard deviation as

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} X_i$$
$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{\mu})^2,$$

and then compute the $(1-\alpha)$ confidence interval for the mean as:

$$\hat{\mu} \pm \eta \frac{\hat{\sigma}}{\sqrt{n}},$$

where η is the $(1-\frac{\alpha}{2})$ quantile of the normal distribution $N_{0,1}$. For all graphs that plot an empirical mean as a function of some underlying variable (i.e., graphs that do not plot an empirical cumulative density function), we have run simulations until the 95% confidence interval is less than ±10% of the empirical mean.

6.2 Suboptimal Anypath Routing

In this section we present two variations on the shortest anypath routing algorithm of Chapter 5. The resulting algorithms do not compute the *shortest* anypath route, but they do compute anypath routes that have lower cost than the shortest single-path route, under the assumption of sometimes-decreasing or always-decreasing anycast link metrics. For each suboptimal algorithm, we also explain how it may offer a reduction in the cost of the relay notification and relay selection mechanisms.

Recall from Chapter 4 (Definition 4.2) that the available relay set (ARS) $A(i)$ of a node i is a subset of i's candidate relay set $C(i)$ containing all nodes that are *available* to forward a given packet, and that the ARS may be different for each successive packet. The ERS policy (Definition 4.4) then chooses, for each packet transmission where $|A(i)| > 1$, the node(s) in $A(i)$ that should effectively relay the packet. The two variations on shortest anypath routing are obtained by using different ERS policies than ERS-best. These policies were briefly mentioned in the previous chapter; their names are *ERS-any* and *ERS-all*.

We also defined in Chapter 4 the remaining path cost (RPC) as

$$R_{iC(i)} = \frac{1}{\mathrm{P}\left(\cup_{k \in C(i)} \mathbf{A}_k\right)} \mathrm{E}\left[f(\mathbf{A})\right], \tag{6.1}$$

where the random vector $\mathbf{A} \in \{0,1\}^{|C(i)|}$ represents the *availability* of nodes in $C(i)$ to relay a packet, ie $\mathbf{A}_k = 1$ if $k \in A(i)$.

The important thing to note in the above definition is that it involves the function $f : \{0,1\}^{|C(i)|} \to \mathbb{R}$ that assigns to a realization of $\mathbf{A}$ the cost of the effective relay that is chosen. Therefore f depends on the ERS, and by consequence so does $R_{iC(i)}$.

As a consequence we must recompute the costs $R_{iC(i)}$ developed in Chapter 4 for each different ERS policy. This dependence of the remaining path cost on the ERS policy shall be illustrated with Example 6.1 (p. 138), once we have given precise definitions to ERS-any and ERS-all.

6.2.1 ERS-any

We have so far considered only the use of the ERS-best policy, that chooses the node $j \in A(i)$ with lowest cost D_j. The new policy for relay selection introduced here is *ERS-any*.

Definition 6.3 (ERS-any)**.** *The* ERS-any *policy selects as effective relay any node j out of $A(i)$. This node j is chosen at random with all nodes in $A(i)$ having equal probability of being selected.*

Remaining path cost with ERS-any

The expression of the remaining path cost with ERS-any is more complicated than with ERS-best, since we must consider individually all possible subsets of $A(i) \subseteq C(i)$. As in Chapter 4, we define q_{ij} to be the probability that node j is an available relay at the outcome of the forwarding phase: $q_{ij} = \mathrm{P}\left(\mathbf{A}_k\right)$. Consider the end of the anycast forwarding phase from node i, and note S ($S \subseteq J$) the available relay set, i.e., the subset of nodes having received the packet from i. Then the remaining path cost is the average cost over nodes in S. We must compute the expected remaining path cost over all $2^{|C(i)|}$ such possible subsets S:

$$\mathrm{E}\left[f(\mathbf{A})\right] = \sum_{S \in 2^{C(i)}} P(S) f(S), \tag{6.2}$$

where $P(S)$ is the probability that the subset of nodes receiving the packet is S

$$P(S) = \prod_{j \in C(i)} (q_{ij} \mathbf{1}_{j \in S} + (1 - q_{ij}) \mathbf{1}_{j \notin S}),$$

and $f(S)$ is the average cost over all nodes in S, because ERS-any chooses a node at random in S:

$$f(S) = \frac{1}{|S|} \sum_{j \in S} D_j.$$

The remaining path cost for ERS-any is then obtained by substituting (6.2) into (6.1), and so in the general case, computing it requires iterating over $2^{|C(i)|}$ possible outcomes of the anycast forwarding phase.

Fortunately, the computation simplifies considerably in the special case where all nodes in $C(i)$ have the same probability of receiving a packet from i. Then, the remaining path cost is simply the average cost of all nodes in $C(i)$, since each node has the same likelihood of being in the random vector $\mathbf{A}$, and the effective relay is chosen uniformly at random out of this vector:

$$R_{iC(i)}^{any} = \frac{1}{n} \sum_{j=1}^{n} D_j, \tag{6.3}$$

where $n = |C(i)|$, and where we have used the superscript ^{any} to emphasize that the above expression is valid only for the policy ERS-any.

This result is in line with the interpretation that with ERS-any, we choose one node uniformly at random from a subset of $C(i)$, and in our special case this subset itself contains each node from $C(i)$ with equal probability. Hence the remaining path cost in this special case is the average over the costs of each node in $C(i)$. We compute this more formally in Appendix 6.8 at the end of this chapter.

RPC for unreliable links with or without ARQ (ETX and E2E metrics)

For always-on networks we have (for the same reason as in Chapter 4) again $q_{ij} = p_{ij}$, and the RPC is the same with or without ARQ, following the same argument as under the ERS-best selection policy.

RPC for asynchronous duty-cycled links

For asynchronous duty-cycling networks, we have (for the same reason as in Chapter 4) again $q_{ij} = p_{hit}$, and so $R^{any}_{iC(i)}$ is the average over D_j as given by (6.3).

We do not show the remaining path costs for synchronous duty cycled links, because with these links anycast forwarding is sender-driven, and so there is no reason to use an ERS other than ERS-best.

Why protocol overhead is lower with ERS-any than ERS-best

Computing anypath routes with ERS-any results in longer shortest anypath routes than with ERS-best. We shall discuss and justify this statement in Section 6.2.3, but for now, suffice to notice that ERS-any sometimes makes a more expensive choice of relay than ERS-best, and thus routes are more expensive overall.

Despite this fact, there is still a good reason to be interested in ERS-any: it is likely that a Relay Arbitration (RA) protocol implementing ERS-any can be carried out at lower cost or complexity than one implementing ERS-best. The reason is that ERS-best imposes a stronger requirement on the RA protocol than ERS-any, since it asks for *one specific* node to be chosen as the effective relay (as opposed to *any* node).

Consider the following RA protocol: each node k in the candidate relay set $C(i)$ that receives a packet from i sets a timer to fire after a random interval $\mathbf{T_k}$. Once this timer fires, node k starts relaying the packet. If node k overhears another node starting to relay the packet before its timer is elapsed, then k knows that a relay is already elected and cancels its timer. If we assume that all nodes in $C(i)$ can overhear each other's transmissions, and with an appropriately chosen distribution for the random timers $\mathbf{T_k}$, then the above scheme will work correctly for ERS-any: exactly one node out of the receivers in $C(i)$ will forward the packet. (This mechanism is somewhat reminiscent of randomized timers used for local recovery in reliable multicast [28] [83].)

The scheme does not work for ERS-best, since it does not take into account the distances D_k from each of the receiving nodes to the destination. To implement ERS-best with random timers, one variation to the above RA protocol would not distribute the $\mathbf{T_k}$ identically, but would rather give bias to nodes with lower D_k such that their timers fire first. However, the $\mathbf{T_k}$ must be spread over an interval large enough that the probability of a node k firing after a node j with $D_j > D_k$ is very small; otherwise the ERS-best policy is often not respected. As a result, the per-hop latency is higher than with ERS-any. This additional latency also has an energy cost, since nodes must keep their radios on to listen for an extended duration.

Another option is to assign *deterministic* time slots to nodes in $C(i)$ by order of increasing D_k. In this case, the ERS-best policy is always correctly applied, but at the cost of added communication overhead. Indeed, nodes must be informed of their assigned timeslot, which requires transmitting at least $\log(|C(i)|!)$ bits to each node (there are $|C(i)|$ timeslots and $|C(i)|$ nodes, resulting in $|C(i)|!$ possible assignments of nodes to slots).

The above argument does not claim to be a rigorous proof that the RA protocol is more expensive for ERS-best than ERS-any, and there can probably be no such proof in a formal sense since we are talking about all possible protocol implementation choices. However we believe that the intuition is sufficiently clear at this point.

What is equally clear is that the assumption used above that all nodes in $C(i)$ can overhear each other's transmissions is a strong one, and that without this assumption, the RA protocol for both ERS-best and ERS-any becomes even more complicated. Fortunately, we are not following the "direct" methodology described in the first section of this chapter, and so we do not immediately jump into designing low-cost RA protocols for ERS-best or ERS-any. We rather turn to a third form of ERS policy, with which the cost of relay arbitration turns out to be nil, even when all nodes in $C(i)$ cannot overhear each other's transmissions.

6.2.2 ERS-all

Our next variation on shortest anypath routing comes from using a third policy for effective relay selection (ERS). It is called *ERS-all*, and is the final ERS policy considered in this dissertation. Unlike ERS-best and ERS-any, ERS-all allows to have *multiple* relays forwarding a same packet. In fact, ERS-all chooses all receivers as effective relay:

Definition 6.4 (ERS-all). *The* ERS-all *policy selects as effective relay every node j in $A(i)$.*

ERS-all can be seen as an extreme case in the sense that we have no ERS policy at all. This has the advantage of completely eliminating the need for a RA protocol. However, with ERS-all we run the risk of having redundant duplicate transmissions, and these may grow out of control over successive hops. It may therefore appear that we need a throttling mechanism to prevent such an explosion, for example by limiting the number of multiple transmissions for each packet injected into a route. At the same time, using ERS-all in conjunction with such a throttling mechanism would be contradictory, since we would be selecting an ERS policy only then to add another layer to counteract its effects. Fortunately we shall see that there are cases of interest where such a throttling mechanism is not needed.

Remaining path cost with ERS-all

The expression of the remaining path cost with ERS-all is similar to that of ERS-any in that we must once again consider individually all possible subsets of $C(i)$ that might receive a packet. The remaining path cost is again obtained by expectation over all $2^{|C(i)|}$ possible subsets S that may receive the packet, and so $\mathrm{E}\left[f(\mathbf{A})\right]$ is defined as in the development for ERS-any (equation (6.2)); $P(S)$ is also obtained as for ERS-any.

The difference with ERS-any is the function $f(S)$. Recall that f assigns to each possible available relay set $A(i) \subseteq C(i)$ the cost of the effective relay that is chosen (under a specific ERS policy). With ERS-all, multiple effective relays can be chosen, and so f simply sums the costs of all nodes that receive the packet:

$$f(S) = \sum_{j \in S} D_j.$$

The remaining path cost for ERS-all is then obtained similarly as for ERS-any, except for this different expression of $f(S)$. Again, computing it requires iterating over $2^{|C(i)|}$ possible outcomes of the anycast forwarding phase, and again, the computation is significantly simplified in the special case where all nodes in $C(i)$ have the same probability of receiving a packet from i. While it is essential to reduce the computation overhead of Bellman-Ford updates, the resulting expression is not particularly useful from an intuitive point of view, and so we leave its computation to Appendix 6.8 at the end of this chapter.

We do however point out two cases of interest, that one can verify using the expression of $R^{all}_{iC(i)}$ computed in the appendix.

- If $q = 1$, the remaining path cost simplifies to $R^{all}_{iC(i)} = \sum_{j \in C(i)} D_j$, corresponding to the observation that when all nodes in $C(i)$ receive a packet, the remaining path cost is the sum of each node's cost.

- If q approaches 0, the remaining path cost simplifies to $R^{all}_{iC(i)} = \frac{1}{n} \sum_{j \in C(i)} D_j$, corresponding to the observation that when only one node (at random) in $C(i)$ receives the packet, then the remaining path cost with ERS-all is identical to that of ERS-any, and indeed also to that of ERS-best.

For the general case where $0 < q < 1$, we will have $\frac{1}{n} \sum_{j \in C(i)} D_j < R^{all}_{iC(i)} < \sum_{j \in C(i)} D_j$.

RPC for always-on networks

For always-on networks we have (for the same reason as in Chapter 4) again $q_{ij} = p_{ij}$, and the RPC is the same with or without ARQ, following the same argument as under the ERS-best selection policy.

RPC for duty-cycled networks with ARQ

For duty-cycled networks, we have (for the same reason as in Chapter 4) again $q_{ij} = p_{hit}$, and so $R_{iC(i)}$ is the sum of D_j for all nodes in $C(i)$ as seen above in the special case for equal q_{ij}'s.

On the dependence of RPC on ERS

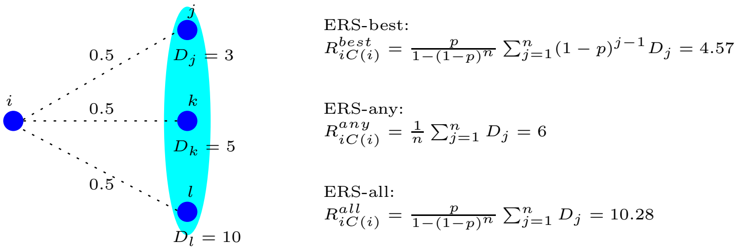

i
0.5
0.5
0.5
j
$D_j = 3$
k
$D_k = 5$
l
$D_l = 10$
ERS-best:
$R^{best}_{iC(i)} = \frac{p}{1-(1-p)^n} \sum_{j=1}^{n} (1-p)^{j-1} D_j = 4.57$
ERS-any:
$R^{any}_{iC(i)} = \frac{1}{n} \sum_{j=1}^{n} D_j = 6$
ERS-all:
$R^{all}_{iC(i)} = \frac{p}{1-(1-p)^n} \sum_{j=1}^{n} D_j = 10.28$

Figure 6.2: Remaining path cost depends on ERS policy. Illustration of Example 6.1.

Example 6.1 (Remaining path cost depends on ERS)**.** We illustrate how the RPC depends on the ERS policy with the example of Figure 6.2. The sender node is on the left, and the candidate relay nodes are on the right, highlighted in a shaded ellipse. Links all have delivery probability $\frac{1}{2}$. With ERS-best, node j is preferred over node k and l, and k is preferred over l (because $D_j < D_k < D_l$), whenever more than one candidate relay receives a packet sent by i. The remaining path cost in this case is computed using equation (4.11) (p. 75) and is 4.57. With ERS-any, there is no preference based on distance from each node to the destination, and one node is chosen at random when there are multiple receivers. The RPC is simply the average of all costs, and is greater than with ERS-best. Finally with ERS-all, every node in the CRS receiving a packet relays it, and so remaining path cost is larger still. Note that the RPC with ERS-all is *not* simply $D_j + D_k + D_l$ because a packet sent by i is not always received by all three nodes in the CRS.

6.2.3 Anypath Route Costs with ERS-any and ERS-all

We now discuss how and why anypath routes are more costly with ERS-any and ERS-all than with ERS-best. We first give a precise meaning to the intuitive notion of "efficiency". In the following definition, we assume that $|C(i)| > 1$ (because with $|C(i)| = 1$ all ERS policies are equivalent) and that $D_j \neq D_k$ for at least one pair of nodes $\{j, k\} \in C(i)$ (because otherwise ERS-any and ERS-best are equivalent).

Definition 6.5 (ERS efficiency)**.** *Consider a node i and candidate relay set $C(i)$, and two ERS policies ERS-x and ERS-y. We say that ERS-x is* more efficient *than ERS-y if*

$$R^x_{iC(i)} < R^y_{iC(i)},$$

and we say that ERS-x and ERS-y are equally efficient *if*

$$R^x_{iC(i)} = R^y_{iC(i)}.$$

Property 6.1. *ERS-best is more efficient than ERS-any, and ERS-any is more efficient than ERS-all.*

This property follows directly from the definitions of remaining path costs for each ERS policy. It is also in line with basic intuition, namely that ERS-best is most efficient because it chooses the next-hop relay with lowest cost to reach the destination, and ERS-all is least efficient because it creates duplicate transmissions which then are forwarded all the way to the destination.

Having defined this notion of ERS efficiency and seen that ERS-best is more efficient than the two others, we can state that for any *given* anypath route, the cost to send a packet across that route will be lower using ERS-best than ERS-any or ERS-all.

Property 6.2. *Consider an anypath route $\mathcal{R}$. Let C^x be the cost (as per Definition 5.4) of this anypath route when using policy ERS-x, and C^y be the cost when using ERS-y.*
If ERS-x is more efficient than ERS-y, then $C^x < C^y$, and if ERS-x and ERS-y are equally efficient, then $C^x = C^y$.

To see this property, consider that the anycast link costs at each node in the route are the same irrespective of the ERS policy, but that the remaining path costs are lower for a more efficient ERS policy.

It is important to observe that the ERS policy is used in two distinct manners. The first is in the routing process: costs computed by the Bellmann-Ford algorithm include the remaining path cost, and this remaining path cost itself depends on the

ERS. As a consequence, the shortest anypath route is in general *not the same when computed under different ERS policies.*

The second way in which the ERS policy is used is for anycast forwarding: when multiple nodes receive a packet, the ERS policy selects the effective relay(s). It is entirely possible to use one ERS policy in the routing process and another for packet forwarding, though this mis-match will cause inefficiencies.

Note that it is not a priori evident from Property 6.2 that ERS-best leads to better routes, because the property applies to the cost of a *given* anypath route using different ERS policies.

Proposition 6.1. *Consider two ERS policies ERS-x and ERS-y. If ERS-x is more efficient than ERS-y, then the shortest anypath route computed with ERS-x has lower or equal cost than the shortest anypath route computed with ERS-y.*

Proof. We prove this with a counter-example. Let $\mathcal{R}^x$ be the shortest anypath route computed using ERS-x, and let $\mathcal{R}^y$ be the shortest anypath route computed using ERS-y. Let us now assume that the cost of $\mathcal{R}^y$ (using ERS-y) is lower than the cost of $\mathcal{R}^x$ (using ERS-x). Note however that from Property 6.2, the cost of $\mathcal{R}^y$ using ERS-y is greater than using ERS-x. Furthermore, because $\mathcal{R}^x$ is the shortest anypath route computed under ERS-x, its cost must be lower than or equal to the cost of any other route using ERS-x, including the route $\mathcal{R}^y$. Therefore the cost of $\mathcal{R}^y$ (using ERS-y) cannot be lower than the cost of $\mathcal{R}^x$ (using ERS-x). ☐

Corollary 6.2. *Shortest anypath routes with ERS-best have lower or equal cost than shortest anypath routes with ERS-any, and shortest anypath routes with ERS-any have lower or equal cost than shortest anypath routes with ERS-all.*

As a final point, one can of course compute a route with a policy ERS-x, and then forward packets using another policy ERS-y. However, this is suboptimal, since the Bellmann iteration would then make decisions based on inaccurate remaining path costs $R_{iC(i)}$. This means for example that computing routes with ERS-best and then forwarding packets with ERS-any is not a sound strategy.

Alternative interpretation

Another way to understand that ERS-any and ERS-all result in longer shortest anypaths than ERS-best is to consider a node i with a singleton candidate relay set $J = \{j\}$. Assume that node j has lowest distance to the destination out of all of node i's neighbors. Now add another node k to the CRS to obtain $J' = \{j, k\}$. We have $D_k > D_j$, and so the remaining path cost with J' will be greater than with J. At the same time, the anycast link cost will be lower (assuming an always-decreasing anycast link cost). Should node j use J' as its candidate relay set? This depends on whether the decrease in anycast link cost is greater than the decrease in remaining path cost:

$$d_{iJ} - d_{iJ'} \overset{?}{>} R_{iJ} - R_{iJ'}$$

Note that anycast link costs are independent of the ERS policy, and therefore so is $d_{iJ} - d_{iJ'}$. However, $R_{iJ'}$ is greater with ERS-any than with ERS-best, because ERS-any will more frequently use k as the next-hop relay. Therefore, ERS-any is less easily able to use an enlarged CRS than ERS-best, and so ERS-any can less frequently take advantage of the corresponding reduction in anycast link cost. In other words, we expect that the shortest anypath when using ERS-any will have smaller candidate relay sets than with ERS-best, and will not be able to take as much advantage of the gains due to an always-decreasing or sometimes-decreasing anycast link cost.

6.3 Anypath Routing Performance

We now examine the performance of anypath routing through a number of simulation experiments. As motivated in the first section of this chapter, we seek to measure the cost of routes only, and consider an ideal implementation in the sense that we do not account for the overhead of relay notification and relay arbitration mechanisms. The goal is to understand the upper limit to the gains that anypath routing can offer, before considering whether or not the overhead of protocol mechanisms will offset these gains in a real implementation.

In this section, we start off with the performance of "optimal" anypath routing, i.e. anypath routing with the ERS-best policy. In the following section, we shall evaluate some suboptimal variants that can be implemented with simpler protocol mechanisms, and see if they give rise to routes that are drastically inferior to optimal anypath routing.

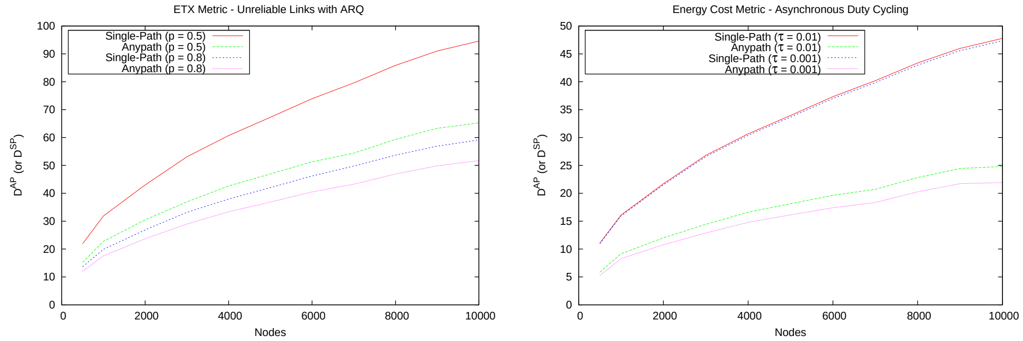
We first look at the performance of anypath routing in a scaling sense; that is, we fix all parameters but one and examine anypath routing performance as network size increases. The primary performance metric of interest here is the *average shortest anypath cost* D^{AP}, which we define as the anypath distance to the destination[3], averaged over all nodes:

$$D^{AP} = \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} D_i,$$

in which D_i is the anypath distance from node i to the destination. (Recall from Def-

---

[3] The destination is randomly chosen for each network realization.



(a) (b)

Figure 6.3: Performance of anypath routing with increasing network size: fixed density, increasing diameter. The length of the shortest (any)path route to a fixed destination is averaged over all nodes in a network, for 10 randomly generated networks at each increasing network size. (a) ETX metric for unreliable links with ARQ. (b) Energy transmission cost for asynchronous duty-cycled links.

inition 5.5 that the *anypath distance* is the distance (or cost) of the shortest anypath route between the two nodes).

We shall use as a comparison point the average shortest single-path cost D^{SP} (to the same destination), that is defined similarly as D^{AP} but using single-path routing. Of course with anycast cost metrics that are always-decreasing, we expect to see $\frac{D^{AP}}{D^{SP}} > 1$, but the question is to see what is the value of this quotient and how it behaves as we scale the network size.

6.3.1 Diameter Scaling

In this first set of simulations, we increase the number of nodes n in a network while keeping average degree ρ fixed, resulting in an increase in network diameter. The shortest single-path routes grow proportionally to this network diameter. What happens to the shortest anypath routes? They will of course increase with network diameter, but it is not clear a priori whether this increase occurs at the same rate, or faster, than for single-path routes. Note immediately that with the type of topology considered, the length of shortest anypath routes cannot increase less rapidly than the diameter, if density remains fixed. In other words, the best behavior we can hope for with diameter scaling is that the shortest anypath routes remain a constant factor shorter than the shortest single-path routes, as opposed to having the anypath routes become less and less short compared to single-path (and possibly even converging).

This can be seen by the following (informal) argument. In a homogeneous network

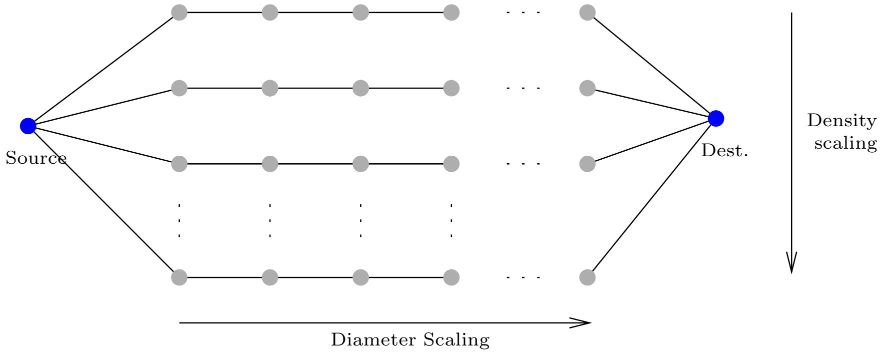


Figure 6.4: Idealized representation of diameter and density scaling. Nodes are displayed in regular grid for clarity; nodes in the simulated networks are randomly distributed.

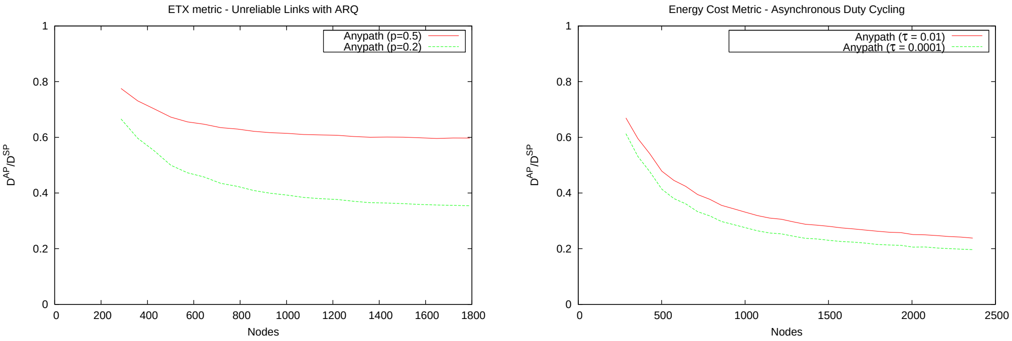
topology such as the one considered here, we can expect that the average candidate relay set size $\frac{1}{|\mathcal{N}|}\sum_{i\in\mathcal{N}}|C(i)|$ is constant if the diameter increases with constant density. Therefore, the anycast forwarding cost at each node will also remain constant. However, the average number of hops that a packet will go through will increase. How fast will this average number of hops increase? In the best case, it will increase as the network diameter, as illustrated in Figure 6.4. The question is now whether Figure 6.4 is representative of a random unit disk graph when it comes to evaluating the scaling behavior of anypath routing.

Always-on networks with ARQ

Figure 6.3(a) shows transmission-count path lengths for always on networks with lossy links; all links have equal delivery probability p. Anypath routes have shorter length at all network sizes, and the ratio $\frac{D^{AP}}{D^{SP}}$ remains constant. For $p = 0.5$, the expected number of transmissions in a route is approximately 45% higher with single-path than anypath. Unsurprisingly, the gap narrows for more reliable links ($p = 0.8$), with single-path being approximately 12% more expensive. With $p = 0.8$, the expected number of per-hop transmissions for unicast is 1.25; the absolute minimum being 1 transmission per hop, there is less room for improvement with anypath.

Asynchronous duty-cycled networks

Figure 6.3(b) shows D^{AP} and $D^{SP}$for asynchronous duty cycling, with two different duty cycles τ. Again, the ratio $\frac{D^{AP}}{D^{SP}}$ is constant. The difference between single-path and anypath routes is further increased in this case, with the cost of single-path routes being just under and just over the double of anypath for duty cycles of $\tau = 1\%$ and



(a) (b)

Figure 6.5: Performance of anypath routing with increasing network sizes: fixed diameter, increasing density. This plot shows the ratio $\frac{D^{AP}}{D^{SP}}$ of average shortest anypath length divided by average shortest single-path length. (a) ETX cost for unreliable links with different link delivery probabilities p. (b) Energy transmission cost for asynchronous duty-cycled links with different duty cycles ρ.

$\tau = 0.1\%$ respectively. The fact that $\frac{D^{AP}}{D^{SP}}$ is greater here than with the previous case can be directly explained by the anycast forwarding gains that we saw in Chapter 4: this gain is higher with asynchronous duty cycling than with always-on ARQ links.

6.3.2 Density Scaling

We now consider networks of increasing number of nodes, but where the *density* grows and diameter remains fixed. Here, the shortest single-path length does not increase with network size[4].

Our intuition here is that $\frac{D^{AP}}{D^{SP}}$ will decrease with network size: the anypath forwarding costs become smaller as $|C(i)|$ increases at each node, while the number of hops traversed by a packet remains constant. This effect is represented in the idealized network of Figure 6.4. The intuition is confirmed by the results of Figure 6.5, which show the ratio $\frac{D^{AP}}{D^{SP}}$ for the ETX metric over unreliable links and for the energy cost metric over asynchronous duty-cycled links. It is instructive to compare the network-wide path gains in these graphs with the gains at the individual node level of Figures 4.6 and 4.10.

These confirm that the overall gain in anypath cost can, to a large extent, be extrapolated from the gains in anycast link costs. This means that at first approxi-

---

[4] In fact, the shortest single-path length slightly decreases, because path dilation effects due to sparsity disappear with increasing density, i.e., the shortest path route becomes closer and closer to a straight line.

mation, a network designer can concentrate on modelling anycast link costs and infer whole-network performance of anypath routing directly from those local costs.

6.4 Suboptimal Anypath Routing Performance

We now study the performance of anypath routing with the suboptimal variants introduced earlier in this chapter. Rather than compute average anypath lengths over an entire network and increase size as we did in the previous section, we now look at networks of *fixed* size, and study the average anypath length for nodes that are increasingly far from the destination. The reason for this slight shift in perspective is twofold. First, we are interested in understanding performance on networks of realistic dimensions and not only in scaling effects. Second, we shall see that examining performance as a function of distance to the destination offers some additional insights in the behavior of different algorithms.

6.4.1 Performance of ERS-any

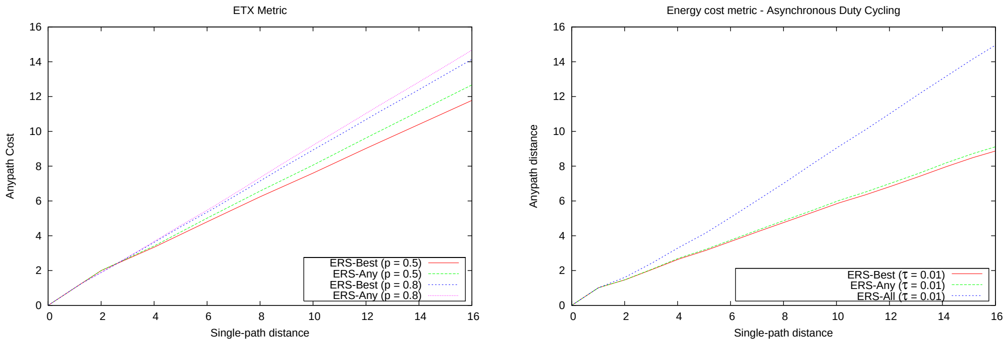
Recall the following tradeoff that was discussed in Section 6.2 of this chapter: for a given CRS J and sender i, the remaining path costs for ERS-best, ERS-any, and ERS-all are ordered as:

$$R_{iJ}^{best} \leq R_{iJ}^{any} \leq R_{iJ}^{all}.$$

Given that the anycast link cost d_{iJ} is the same irrespective of the ERS, this means that the Bellmann equation will find optimal candidate relay sets that are smaller with ERS-any and ERS-all than with ERS-best. Consequently, the routes computed with ERS-best will have lower cost than those computed with the other ERS policies, because they will have lower anycast link costs.

We compare ERS-best and ERS-any in Figure 6.6. The first plot shows results for always-on networks with a transmission-count metric. As in the scaling simulations, all links have same delivery probability p. For $p = 0.5$, ERS-any requires at most 10% more packet transmissions than ERS-best, and for $p = 0.8$, this difference falls below 5%. From a practical design standpoint, this indicates that if the difference in protocol overhead to perform relay arbitration with ERS-best compared to ERS-any is greater than 10% of the cost to transmit a packet, then ERS-any deserves serious consideration.

We now turn to the comparison of ERS-best and ERS-any for asynchronous duty cycling in Figure 6.6(b). Here, ERS-best continues to out-perform ERS-any, but the margin is sharply reduced. This margin in fact becomes negligible when the duty cycle is 0.001. Again, this indicates from a practical standpoint that ERS-any



(a) (b)

Figure 6.6: Comparison of average shortest anypath cost D^{AP} for different ERS policies. (a) ETX metric for unreliable links with ARQ. (b) Energy transmission cost for asynchronous duty-cycled links.

should be considered when designing an anypath protocol for asynchronous duty-cycled networks.

Relative performance of ERS-any depends on link model

Why is the relative performance of ERS-best and ERS-any significantly closer with the energy cost metric for asynchronous duty cycling than with using the ETX metric? A natural explanation would be that the distance "spread" between nodes in a candidate relay set is greater in one model than another. Consider a node i, and define $C_1(i)$ to be the candidate relay set of a node i in a network using the ETX metric, and define $C_2(i)$ to be the same node's CRS with asynchronous duty-cycling and an energy-cost metric. Let us define the distance spread s_1 to be the sample variance of the candidate relays' distances to the destination:

$$s_1 = \frac{1}{|C_1(i)|} \sum_{j \in C_1(i)} (D_j - m_1)^2,$$

where m_1 is the average distance over nodes in $C_1(i)$, and define s_2 similarly for $C_2(i)$. If $s_1 >> s_2$, then the gap between ERS-best and ERS-any will be larger when using $C_2(i)$ as candidate relay set than when using $C_1(i)$. The converse is also true, namely that if all candidates are at same distance from the destination, then there is no difference at all between ERS-any and ERS-best.

Can a difference in delay spread be a valid explanation for the difference in relative performance of ERS-any and ERS-best between the two graphs of Figure 6.6? Note that protocols are compared in paired experiments, and so any difference in spread cannot be due to differing network topologies. So if there is a difference in delay spread, it must be attributable to the anypath routing algorithm itself. However the following argument shows that distance spread should if anything be *greater* with the energy cost metric than with transmission count. Recall from Chapter 4 (Figure 4.6(b), p. 73 and Figure 4.10(a), p. 81) that ALC gain for the energy cost with asynchronous duty cycling increases *more rapidly* with $|C(i)|$ than the ALC gain for the ETX metric. Therefore under the former metric the Bellman-Ford iteration is able to select a CRS with a larger distance spread, since there is a larger corresponding ALC decrease to offset an additional node with higher distance.

ERS-any close to ERS-best with small available relay sets

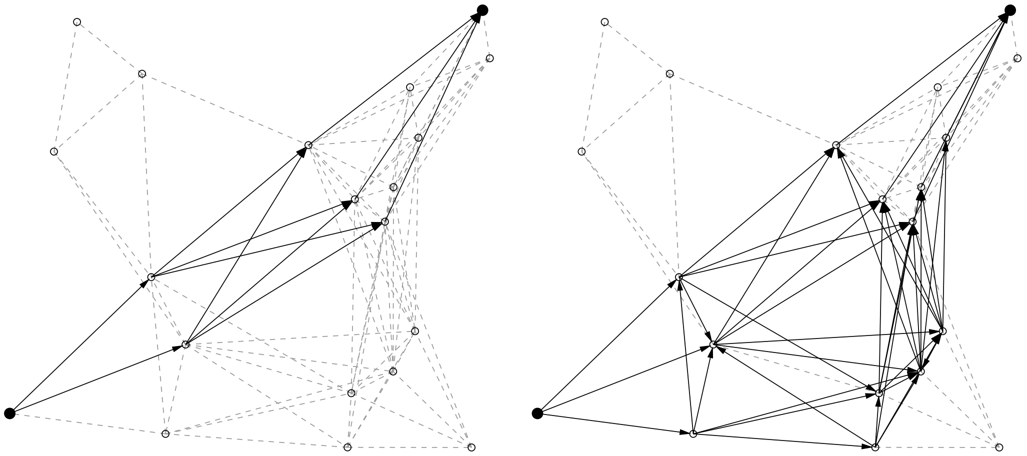
We must therefore seek another explanation for this difference in relative performance of ERS-best and ERS-any for different cost metrics. What should really surprise us in Figure 6.6 is not the relative difference between the two plots, but simply the result of plot (b) alone, showing that there is only a minute difference between ERS-best and ERS-any in the case of asynchronous duty cycling.

The explanation for the near indifference to the choice of ERS-best or ERS-any in Figure 6.6(b) is the following. Recall that our definition of anycast transmission cost (4.13) used the (normalized) preamble length λ minimizing transmission cost, and that this minimum was reached for $\lambda < 1$. Specifically Figure 4.9 showed that the minimum was reached for very small values $\lambda < 0.1$. With such small preamble lengths, the probability of hitting *multiple* nodes in a single transmission is small. This means that the available relay set $A(i)$ rarely has multiple nodes, and so the difference between ERS-best and ERS-any rarely comes into play. As a result, the anypath routing algorithm can choose candidate relay sets that are (almost) as large with ERS-any as with ERS-best, giving routes with similar cost.

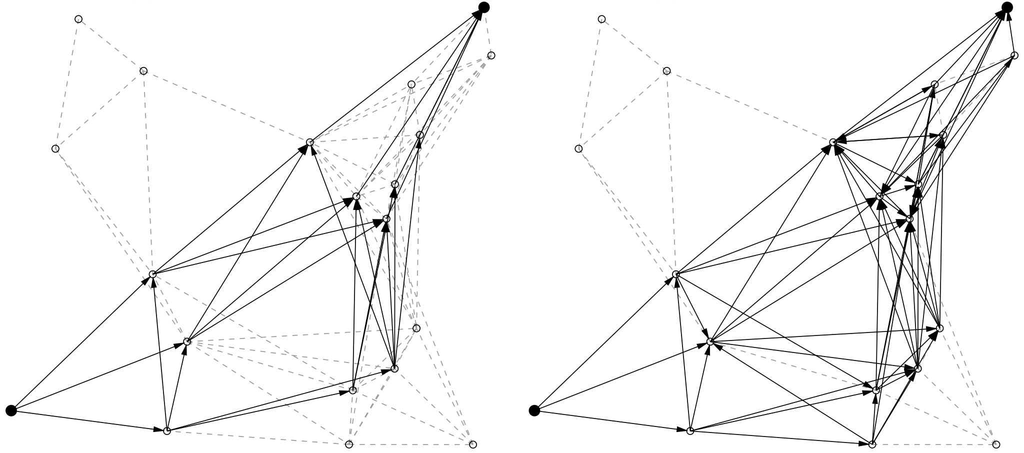
In comparison, the probability in Figure 6.6(a) that $A(i)$ contains multiple candidate relays is significantly larger, since we have $p = 0.5$ and $p = 0.8$. Therefore in that model, the ERS policy more often is called upon to choose an effective relay, and the suboptimal choices made by ERS-all force the anypath routing algorithm to choose smaller candidate relay sets than with ERS-best. This is illustrated in Figure 6.7.

6.4.2 Performance of ERS-all

Moving down the ERS "food chain", we now study the performance of ERS-all. From the lesson of the previous paragraphs, we can make the following hypothesis: if the



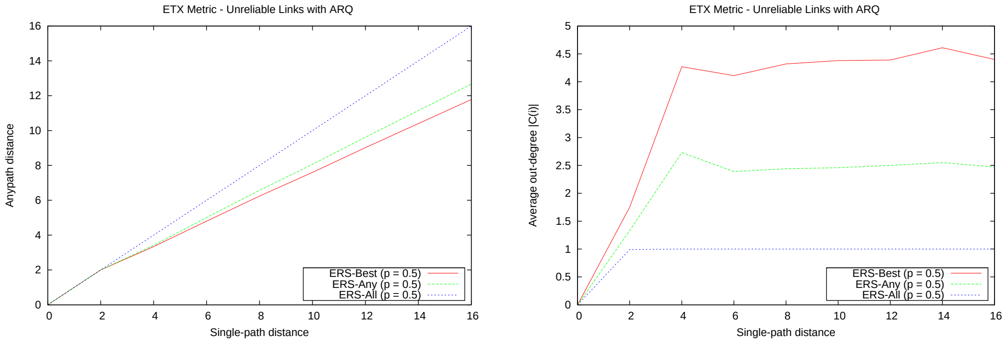
(a) ERS-any, $p = 0.7$ (a) ERS-best, $p = 0.7$



(a) ERS-any, $p = 0.3$ (a) ERS-best, $p = 0.3$

Figure 6.7: Comparison of shortest anypath routes computed with ERS-any and ERS-best. The cost model is ETX with unreliable ARQ links. Routes are computed in simulation using the anypath Bellman-Ford algorithm. Top: All links have packet delivery probability $p = 0.7$. The shortest anypath route has much smaller CRS sizes with ERS-any than with ERS-best, because the likelihood of having multiple nodes in $A(i)$ (and possibly choosing a suboptimal relay) is high. Bottom: Links have packet delivery probability $p = 0.3$. Anypath routing can use larger CRS sizes with ERS-all, since the probability that $|A(i)| > 1$ is reduced. ERS-best also behaves differently for each value of p. With $p = 0.3$, the algorithm takes larger candidate relay sets than with $p = 0.7$, because the anycast forwarding gain $G(d_{iJ})$ grows faster with $|J|$ for small p.

probability of each individual node in the CRS receiving a packet is very small, then the performance of ERS-all will be less degraded relative to ERS-any than with high



(a) (b)

Figure 6.8: Comparison of ERS-all with ERS-best and ERS-any for transmission-count metric with link delivery probabilities $p = 0.5$. (a) ERS-all has worse performance, but does not lead to an explosion in transmissions, giving same cost routes as single-path. (b) This explosion is avoided because with ERS-all, the anypath algorithm always takes $|C(i)| = 1$, resulting effectively in single-path routes.

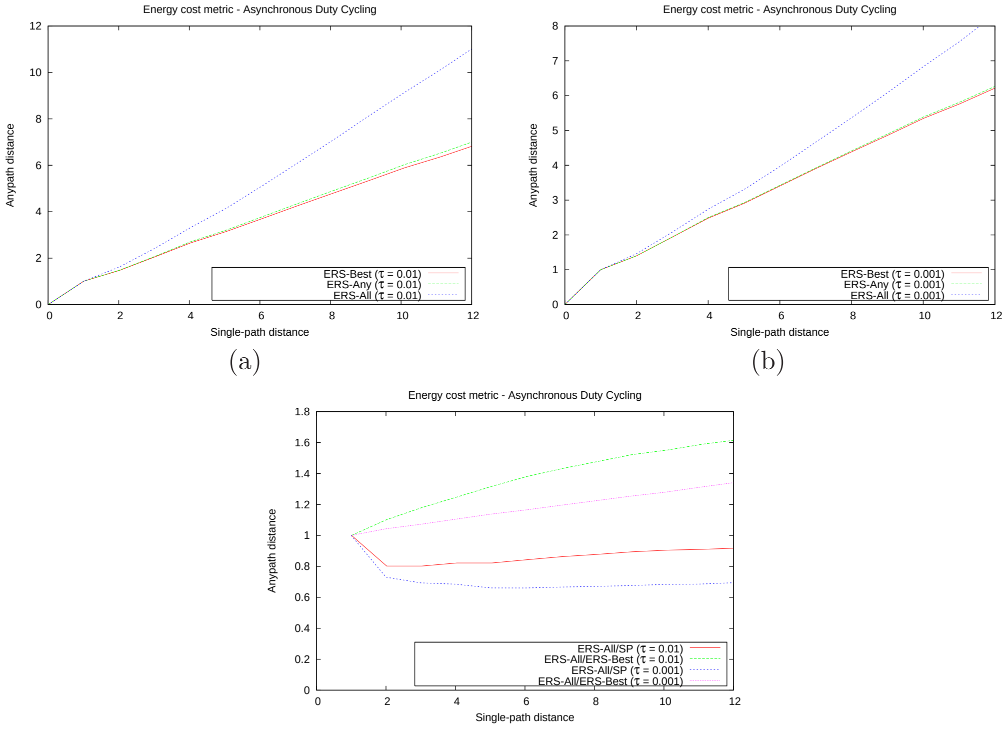
receiving probabilities. The argument follows along the same lines as the previous one: if the probability of $A(i)$ containing multiple nodes is very low, then the ERS-all policy will rarely let duplicate packets go through, and so performance will not suffer.

If this probability is large, then it appears that ERS-all may frequently let multiple packets through, and so performance will suffer. In fact, will we get an explosion of transmissions with the number of duplicate copies growing rapidly at each hop?

ERS-all with large available relay sets

We first explore the question formulated above, to see if ERS-all results in a transmission explosion when the probability of having multiple nodes in $A(i)$ is large. We compare in Figure 6.8 ERS-all and ERS-any for the ETX metric with $p = 0.5$. As expected, ERS-all performs less well than ERS-any, but it does not lead to an explosion in transmissions. In fact, ERS-all provides exactly the same performance as single-path routing. This illustrates how our anypath routing algorithm *takes into account* the ERS-policy by integrating the remaining path cost in the overall cost computation at each node.

With ERS-all and $p = 0.5$, the increase in remaining path cost from adding a node to the CRS outweighs the decrease in anycast forwarding cost, and so the anypath algorithm simply uses singleton relay sets, resulting effectively in single-path routing. We illustrate this in 6.8(b), by showing the *average outdegree* for nodes as a function of their distance to the destination. (The outdegree for a node i is simply the size $|C(i)|$



(a) (b)

(c)

Figure 6.9: Comparison of ERS-all with ERS-best and ERS-any for asynchronous duty cycling with energy-cost metric. The gap between ERS-all and ERS-best is wider for duty cycle $\tau_1 = 0.01$ (a) than for $\tau_2 = 0.001$ (b). (c) The ratio of ERS-all to ERS-best route lengths and of ERS-all to single-path route lengths, is not constant, showing that performance of ERS-all worsens for increasingly long routes.

of that node's candidate relay set). With ERS-all, the anypath routing algorithm uses $|C(i)| = 1$, because the cost of duplicate transmissions is too high to offset the reduction in forwarding cost with multiple candidate relays. This figure also illustrates how the algorithm adapts to the difference between ERS-best and ERS-any, with larger $|C(i)|$ sizes for the former than for the latter.

Of course the fact that ERS-all does not blow up in this case is good news, but should not obscure the fact that it underperforms ERS-any and ERS-best. Therefore, when multiple nodes frequently receive a packet, a protocol designer should avoid ERS-all or else anypath routes will simply be shunned by the anypath routing protocol.

ERS-all with small available relay sets

Having confirmed with Figure 6.8 that ERS-all forces anypath routing to use single-path routes when $|A(i)|$ frequently is greater than 1, we return to the first hypothesis formulated at the start of this section: if the probability of each individual node receiving a packet is very small, then the performance of ERS-all will not be significantly degraded relative to ERS-any. We verify this hypothesis by returning to asynchronous duty cycling. We consider two duty cycles $\tau_1 = 0.01$ and $\tau_2 = 0.001$. Using (4.13), we compute numerically that the optimal preamble lengths are $\lambda_1 = 0.067$ and $\lambda_2 = 0.022$. How often will ERS-all result in duplicate packets being transmitted toward the destination? This is equal to the probability that $|A(i)| > 1$, or equivalently that more than one node receives the packet in the last round of the anycast forwarding phase. Recall that with asynchronous duty cycling, the sender retransmits until any one node in $C(i)$ receives the packet. A duplicate packet forwarding with ERS-all will thus happen if any node(s) out of $|C(i)|-1$ nodes receive the packet, conditional on the event that the other node (not in those $|C(i)|-1$ nodes) has received it. Noting $n = |C(i)|$, the expression of this probability as a function of λ is:

$$p(\lambda) = 1 - (1-\lambda)^{(n-1)}.$$

Evaluating this formula for $n = 4$ (which is the average CRS size in Figure 6.8), we get that $p(\lambda_1) = 0.19$ and $p(\lambda_2) = 0.06$.

This computation indicates that, at least for the duty cycle $\tau_2 = 0.001$, the probability of $|A(i)| > 1$ is low enough that anypath routing should be able to select non-singleton candidate relay sets with ERS-all, and so that we should see at least *some* gains over single-path routing. The remaining question is how far the gap between ERS-all and ERS-any or ERS-best will be, and we turn to Figure 6.9 which shows anypath route performance for τ_1 and τ_2 , again as a function of single-path distances.

As expected, ERS-all underperforms ERS-any more severely with $\tau_1 = 0.01$ than with $\tau_1 = 0.01$, and in fact is only marginally better than single-path routing in this case. For $\tau_2 = 0.001$, the gap is less wide, but still remains significant (up to 40%). With such a difference, selecting ERS-all over ERS-any can only be worthwhile if the overhead of relay arbitration is large. This difference may appear surprising given the values for $p(\lambda_2)$ computed above: with a probability of only 0.06 of having duplicate relays, ERS-all still underperforms by 40%.

Performance of ERS-all depends on distance

In order to explain this surprising observation, let us first remark that the gap between ERS-all and ERS-any widens with increasing distance to the destination, as Figure 6.9(c) demonstrates. This is unexpected, given that in all previous plots we had gains that appeared to be constant with increasing distance (the first hop notwithstanding, where $C(i)$ is the destination itself and so always of unit size).

However it turns out that there is a natural explanation. The intuition is the following: if a packet is many hops away from the destination, the cost of having it duplicated and forwarded redundantly by multiple relays is much higher than if the same packet is only one or two hops away. In some sense, the "risk penalty" associated with ERS-all forwarding redundant copies is much greater at distant nodes than nearby nodes. Therefore with ERS-all faraway nodes must reduce their CRS size to compensate for the increased penalty associated with duplicate relays. This effect is illustrated with the simulated network of Figure 6.10.

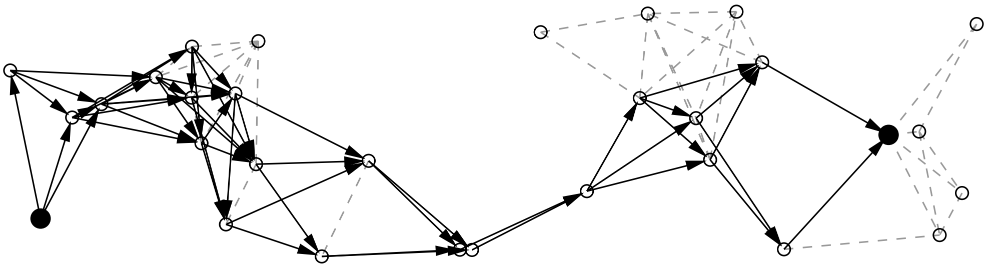
The average outdegrees are plotted in Figure 6.11. While $|C(i)|$ remains stable at increasing distances for ERS-best and ERS-any, it decreases for ERS-all. The decrease is faster for τ_1 than τ_2, as the likelihood of multiple receivers is greater, and so the anypath algorithm clamps down on CRS sizes more rapidly. For either value of τ the average out-degree will eventually converge at 1, resulting in routes that are effectively single-path in their faraway (from the destination) portions and anypath in their close portions.

ERS-all reduces CRS sizes at increasing distances

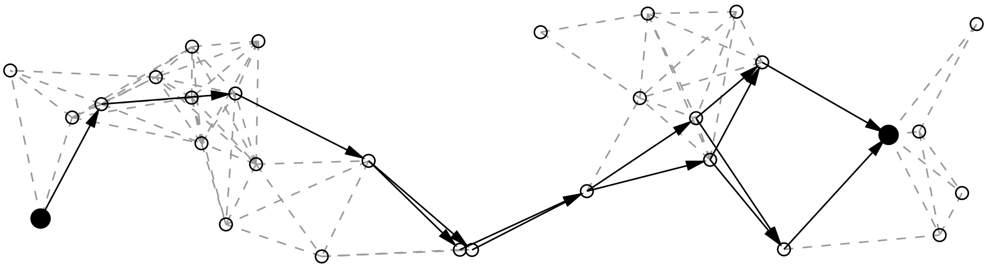
It is somewhat remarkable that the anypath algorithm can integrate this tradeoff of having multiple relays and progressively restrict CRS sizes as it moves away from the destination. We illustrate how this is made possible with the following example.

Example 6.2 ("Risk penalty" of ERS-all increases with distance)**.** Consider a node i that can select as its candidate relay set either $C_1(i) = \{j\}$, or $C_2(i) = \{j, k\}$. Let us assume that both neighbors j and k have same distance to the destination $D_j = D_k$. The remaining path cost with $C_1(i)$ is $R_{iC_1(i)} = D_j$. To compute the $R_{iC_2(i)}$, let us assume that the probability of both nodes forwarding the packet from i is 0.06 (same value as $p(\lambda_2)$ above). The remaining path cost with $C_2(i)$ is then

$$R_{iC_1(i)} = 0.94 \cdot D_j + 0.06 \cdot (D_j + D_k) = 1.06 \cdot D_j$$



(a) ERS-best



(b) ERS-all

Figure 6.10: Comparison of shortest anypath routes computed with ERS-all and ERS-best. The cost model is transmission energy with asynchronous duty cycling. The routes are computed in simulation using the anypath Bellman-Ford algorithm. In the shortest anypath route with ERS-all, the CRS size is progressively decreased to 1 moving away from the destination, since the cost of duplicate packets is much higher there. Within two or three hops of the destination, the shortest anypath ERS-all route uses larger candidate relay sets, and becomes quite similar to the shortest anypath ERS-any route.

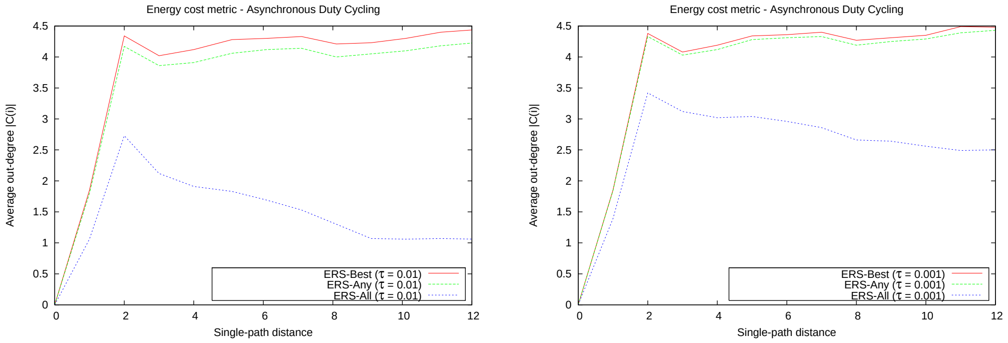
The anycast link cost depends on the chosen metric. Let us assume that we have

$$d_{iC_1(i)} = 1, \quad \text{and} \quad d_{iC_2(i)} = \frac{2}{3}.$$

Node i will take as candidate relay set the set minimizing the sum $d_{iC(i)} + R_{iC(i)}$. It therefore selects:

$$\begin{array}{ll} C_1(i), & \text{if} \;\; 0.06 \cdot D_j \geq \frac{1}{3} \\ C_2(i), & \text{if} \;\; 0.06 \cdot D_j < \frac{1}{3}, \end{array}$$

illustrating how the anypath algorithm will choose the smaller CRS as the distance to the destination increases.



(a) $\tau = 0.01$ (b) $\tau = 0.001$

Figure 6.11: Average out-degrees for asynchronous duty cycling. With ERS-all, $|C(i)|$ decreases with distance to the destination, explaining why ERS-all's performance worsens for long routes.

6.5 Anypath Routing with Single-Path Metrics

We explained in Chapter 1 that existing approaches using single-path metrics could be improved upon, because these approaches base their forwarding decisions on the *single-path* distance of nodes to the destination, and the ordering of nodes by single-path distance is generally not the same as by anypath distance. We have also shown how to compute shortest anypath routes, that are the best anypath routes for a given network and cost metric. By how much do these routes improve over routes driven by single-path, such as those computed by ExOR [9]?

In the following, *AP* routes are shortest anypath routes. *SP-AP* routes are ExOR like anypath routes obtained using a single-path metric: nodes run a standard single-path algorithm and take as candidate relays all neighbors with lower single-path cost to reach the destination.

It is important to point out that the gap between AP and SP-AP routes is largely determined by network topology. As we have previously remarked, SP-AP routes are inferior to AP because their distance ordering of nodes is generated by a metric that at best approximates the true anypath distance of nodes to the destination. Note that it is the difference between *orderings* that deteriorates SP-AP's performance, and not the difference between distance metrics. For example, if SP-AP distances are equal to AP distances, multiplied by a constant factor, then the orderings are the same and SP-AP finds the shortest-path routes. We illustrate this with a lattice network in Figure 6.12. The North and East neighbors of each node are closer in single-path distance to the destination, and so are the candidate relays in a SP-AP route. The

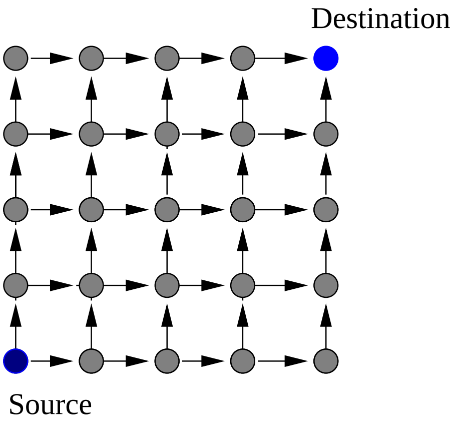


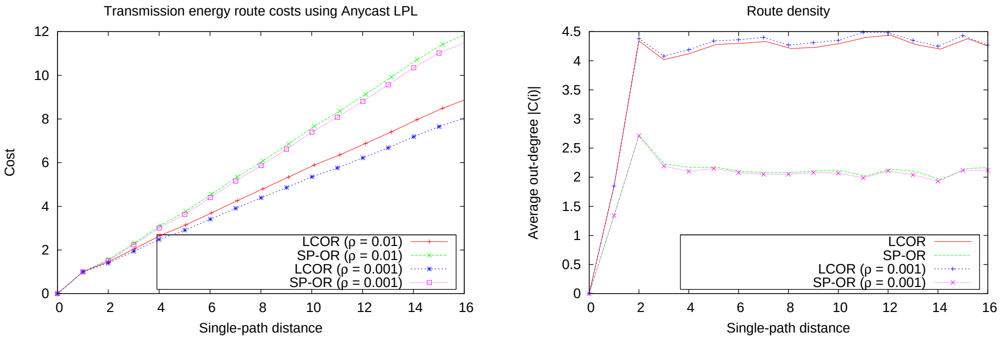
Figure 6.12: Example network where anypath routes determined by single-path metrics are identical to shortest anypath routes. In single-path distance, the North and East neighbors of each node are closer to the destination, and so would be used by anypath routing that uses single-path metrics (e.g., ExOR). The resulting route is equal to the shortest anypath route. This convergence of single-path metrics with shortest anypath metrics is due to the *regularity* of the network; contrast this network for example with that of Fig. 1.4 in Chapter 1.

same is true under anypath distances, and so the SP-AP route is equal to the AP route in this example.

This example, when contrasted with Figure 1.4 (p. 1.4), suggests that *regularity* or homogeneity of a network are key to the relative perfomance of SP-AP versus AP. While the networks that we simulate in this chapter are not lattices, they are still fairly homogeneous, with nodes being uniformly distributed in the plane. We turn to the simulation results of Figure 6.13 that plot the performance of both routes using the same simulation setup as in Figure 6.11. The cost of the SP-AP routes is approximately 40% higher than that of shortest anypath routes (AP); nodes in AP routes have about 4 candidate relays, in comparison with 2 for SP-AP routes.

6.6 Robustness and Stability

The previous sections in this chapter focused on the effective cost of anypath routes and compared this cost for different forms of anypath routing. We saw that anypath route costs were lower by a significant factor for a multitude of metrics including latency, energy, and expected number of transmissions. These gains are significant, and we believe that they alone provide a strong motivation for the use of anypath routing. However, there are additional aspects of performance that we have not



(a) (b)

Figure 6.13: Comparison of AP and SP-AP routes. (a) Comparison of average shortest anypath costs (b) Average out-degrees.

considered so far, and that are of particular relevance to distributed implementations. These are *robustness* and *stability*, and are the object of study in this section.

An important assumption was made implicitly up to this point: that the routing protocol has *perfect* knowledge of the network topology over which it computes routes and forwards packets. In other words, the inputs to the routing algorithm are the exact network topology, and not some approximation thereof. This assumption is a standard one, and necessary to define and reason about routing algorithms and the routes that they compute. However, most networks (and wireless networks in particular) are uncertain and time-varying objects, and so this assumption of perfect knowledge does not always reflect the practical reality of network protocols.

Causes of imperfect knowledge

There are two equivalent ways to see how protocols must in practice work with an imperfect representation of network topology. The first is that the network changes over time. In this case, whatever mechanism is used to track network topology (for example, using periodic beacons to keep track of neighbors as described in Section 6.1) will invariably *lag* behind the actual topology, and the routing protocol will have at its disposal a representation of topology that is (at least partially) out of date. This lag may in practice be negligible, or it may be very large: its extent depends on the rate of network change relative to rate at which link estimation and neighbor tracking are carried out.

The second interpretation is that the network is static, but the network protocol uses imperfect estimators to track link delivery probabilities and neighbors. Assuming the presence of such imperfect building blocks may appear surprising, and one immediate reaction might be that these should first be fixed before building a routing protocol on top of them. However, experience shows that link estimation is a very hard problem [3, 32, 114], especially without having a continuous stream of samples (i.e., transmissions of data packets or channel probe packets) to feed the estimator. This means that the estimates of $\hat{p}_{ij}$ used by the protocol are not exactly equal to the underlying p_{ij}. With E2E or ETX metrics the link distances will therefore be incorrect; with link filtering the protocol will then sometimes exclude links that should be included and vice-versa.

Effects of imperfect knowledge

Whichever of the two interpretations above best explains the presence of imperfect information in practice, the result is the same: in a real implementation, we are in effect routing with an *imperfect* or *noisy* representation of the network's current topology.

For this reason, a routing protocol will sometimes provide routes that are *not* the shortest path routes in the current topology; no protocol will in practice perform exactly at its theoretical optimum. Different protocols may be more or less sensitive to the use of imperfect knowledge, and their performance may degrade more or less rapidly. A complete evaluation of anypath routing must therefore assess how sensitive (or not) it is to noisy topology estimates.

Consider two routing algorithms X and Y, and assume that "X is less sensitive than Y" (for some informal definition of sensitivity, based for example on the notions of robustness and stability defined below). There are two ways to map this into a performance gain for X. The first and more direct way is that in the face of equally inaccurate link estimates, X will perform closer to its theoretical optimum than will Y. The second way is that we can have more inaccurate link estimates for X than Y (and for example reduce in the number of link estimation beacons transmitted) while having both protocols perform at the same point relative to their respective optima.

6.6.1 Time-varying Network Models

We use three very simple processes as models of how a network may change over time. Each model is driven by a single parameter that controls how fast the network varies. We call this the *variation parameter* p_v.

The first is *node-based*, in that it considers nodes being added to and removed from the network. This can be seen as a (highly idealized) representation of a network in which some nodes may at times be unreachable due for example to software crashes, hardware failures, or energy depletion. The second and third are *link-based*, considering links that are added to or removed from the network, and links that change cost over time. These can be seen as a representation (again idealized) of the continuous link fluctuations that take place in wireless networks.

Node addition/deletion model

We start off with a *base graph* $\mathcal{G}$. At each time index t we have a new realization $\mathcal{G}_t$ of the graph. This model has a single parameter p_v, which is the probability of node deletion: each node in $\mathcal{G}$ is present in $\mathcal{G}_t$ with probability p_v, except for the destination which is never removed. All links that are incident to a deleted node are also removed. Note that while we define this model in terms of node deletions only, the same nodes are typically not deleted in two consecutive graphs, and therefore two graphs $\mathcal{G}_t$ and $\mathcal{G}_s$ will usually each contain nodes that the other does not. In other words, some nodes are added as well as removed when going from a graph $\mathcal{G}_t$ to $\mathcal{G}_s$.

Link fluctuation model

The link fluctuation model applies to networks with lossy links, and modifies the link delivery probabilities p_{ij}. As previously, we start off with a *base graph* $\mathcal{G}$ and define a sequence of graphs $\{\mathcal{G}_t\}$. Each successive graph contains the same set of links and nodes as $\mathcal{G}$. The difference is that each link delivery probability p_{ij} is either increased or decreased by a small increment x (subject to the constraint that $p_{ij} \in [0, 1]$). The increment is uniformly distributed over an interval of size p_v, and so this model is also driven by the single parameter p_v. The link fluctuation model is more succinctly described as:

$$(p_{ij})_t := p_{ij} + \mathbf{X}, \qquad \text{where } \mathbf{X} \sim U\left[-\frac{p_v}{2}, \frac{p_v}{2}\right]$$

where $(p_{ij})_t$ is the link delivery probability between nodes i and j in $\{\mathcal{G}_t\}$, and the above update rule is applied to all links in $\mathcal{G}$.

Link addition/deletion model

The final model is also link-centric and varies graphs by removing links from the base graph $\mathcal{G}$. Each link is removed with probability p_v. As with the node addition/deletion model, links are both added and removed when comparing two graphs $\mathcal{G}_t$ and $\mathcal{G}_s$. This model applies to both networks with lossy links and networks with reliable links, unlike the fluctuation model that does not serve with reliable links.

Memoryless versus evolving graphs

We note that all models proposed here are memoryless when conditioned on the base graph $\mathcal{G}$, that is, the probability of observing a given graph $\mathcal{G}_t$ at time t remains the same irrespective of the outcomes of the preceding graphs $\mathcal{G}_t$, $s < t$. Another approach would have been to define a series of evolving graphs where each successive realization $\mathcal{G}_t$ is a function of its predecessor $\mathcal{G}_{t-1}$, hence giving a graph evolution process with memory. Our choice of the memoryless model is motivated by the fact that we are considering static wireless networks, and so while these change over time (due to channel fluctuations, node failures, etc), we do not expect them to truly "evolve" in a way that the topology at a given point in the future becomes completely decorrelated from the topology at the present time. A natural way to view this is that the network is the sum of an underlying deterministic object (the nodes and their locations) with a continuous and stationary "noise", where this noise may be a combination of link and node perturbations, additions, and removals.

6.6.2 Quantifying Robustness and Stability

Being now armed with some models of time-varying networks, we must define what is meant by robustness and stability, in order to evaluate how various algorithms perform in the face of network change and with inaccurate knowledge of topology. The concept of robustness is often wielded somewhat loosely in the study of computer systems, and even sometimes appears to be used as a subjective, aesthentic criterion; we seek rather to give it a quantitative definition.

The goal is to see, for a given time-varying network model and variation parameter p_v, how quickly routes computed on a network $\mathcal{G}_s$ become suboptimal on another network $\mathcal{G}_t$ for different routing algorithms. We are therefore interested in seeing how well (or poorly) the routes computed on network $\mathcal{G}_s$ will perform when used on a similar, but not identical, network $\mathcal{G}_t$. First, we must define more precisely what we mean by "using a route on a given network".

Definition 6.6 (Anypath route projection)**.** *Consider a route $\mathcal{R} = (\mathcal{N}_R, \mathcal{A}_R)$ defined as a set of nodes and directed edges. This route can be either single-path or anypath. The* projection *of this route on a graph $\mathcal{G} = (\mathcal{N}_G, \mathcal{A}_G)$ is another route, defined as:*

$$(\mathbf{proj}(\mathcal{R}, \mathcal{G})) = (\mathcal{N}_R \cap \mathcal{N}_G, \mathcal{A}_R \cap \mathcal{A}_G).$$

An example of route projection is shown in Figure 6.14.

We now describe our measures of stability and robustness. In the following, $\mathcal{R}_t$ is the shortest (any)path route from a node i to the destination in $\mathcal{G}_t$, and $\mathcal{R}_s$ is the shortest (any)path route in $\mathcal{G}_t$. A route computed on $\mathcal{G}_s$ can be suboptimal in $\mathcal{G}_t$ in one of three ways:

1. *The route becomes disconnected.* A route is disconnected when it contains no path from the source to the destination. A single-path route is disconnected as soon as any of its links or nodes are deleted; disconnection of an anypath route usually requires multiple nodes or links to be cut (except of course if the anypath route goes through a “bottleneck” node with a singleton candidate relay set, in which case a single deletion suffices to disconnect it). This is a binary characterization, that considers simply whether or not a route is still functional.

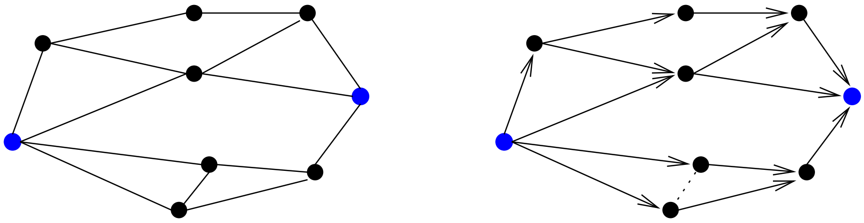
2. *The route remains connected, but has dead ends.* Seeing how easily a route gets disconnected gives a first-order assessment of the resilience of different routing strategies in the face of changing network topologies. While it is clearly preferable for an anypath to remain connected, performance can still be affected if it contains *dead ends*. A dead end is a node i such that for each node j in $C(i)$, either the link (i, j) is disconnected, or the node j is deleted. Note that for single-path, the disconnection and the presence of a dead end(s) are equivalent criterion.

   With anypath, a packet arriving at a dead end must either be dropped or returned upstream in the route. Or, to avoid such situations altogether, an anypath protocol must have a *pruning* mechanism to tear down paths leading to a dead end. A fair comparison of resilience between anypath and single-path should therefore take into account the presence of dead ends in addition to complete disconnection, since an anypath route can remain connected but suffer from dead ends.

3. *The route remains connected, but is dilated.* Even if a route $\mathcal{R}_s$ remains connected when projected on $\mathcal{G}_t$, it is possible that it is not the shortest route in $\mathcal{G}_t$ even if it was the shortest route in $\mathcal{G}_s$. In other words, it may happen that a new shortcut link appears in $\mathcal{G}_t$ that reduces the length of the shortest (any-) path route between a pair of nodes, or that a change in p_{ij}’s cause the shortest route in $\mathcal{G}_t$ to be different than the projection of $\mathcal{R}_s$ on $\mathcal{G}_t$. In this case, we are interested to see how well a route that was computed for a topology $\mathcal{G}_t$ performs when used over the new topology, using as a benchmark the shortest path route in $\mathcal{G}_s$.

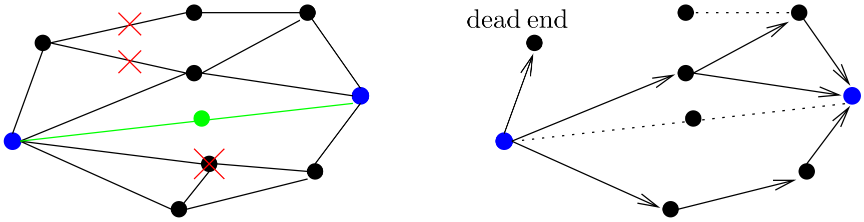
(a) Graph $\mathcal{G}_t$

(b) Shortest anypath route $\mathcal{R}_t$



(c) Graph $\mathcal{G}_s$

(d) $\mathbf{proj}(\mathcal{R}_t, \mathcal{G}_s)$



(e) Shortest anypath route $\mathcal{R}_s$

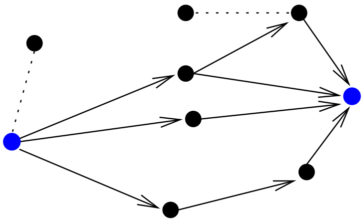


Figure 6.14: Example of a route projection. (a) Graph $\mathcal{G}_t$, with a given source (left) and destination (right) nodes in blue. (b) Shortest anypath route $\mathcal{R}_t$ in $\mathcal{G}_t$. (c) Second realization $\mathcal{G}_s$, with two links and one node removed (red), two links and one node added (green). (d) The projection of $\mathcal{R}_t$ on $\mathcal{G}_s$, has one dead end. (e) The shortest path $\mathcal{R}_s$ on $\mathcal{G}_s$ takes advantage of the added (green) node and links; it has lower cost than $\mathbf{proj}(\mathcal{R}_t, \mathcal{G}_s)$.

Comparing with single-path

In practice, some single-path routing protocols for wireless networks keep track of *back-up* next hops, that allow the protocol to failover to an alternate path if the primary path fails. This is a practical way to increase the robustness of single-path routing to network changes, and adding this failover capability requires little overhead beyond the storage of multiple next-hop nodes in a forwarding table.

It would be unfair to compare the robustness of anypath routing exclusively with "standard" single-path routing, and not consider the improvement described above. We shall therefore consider both single-path (SP) routing and single-path with k-node failover (SP-k). This parameter k represents the maximum number of additional nodes that the protocol maintains in its routing table as a backup, and hence SP-0 is equivalent to SP. Of course, single-path with failover cannot take any neighbor as a backup next hop, or else loops will form. A node can keep as backup any neighbor with lower distance to the destination than itself, under the constraint of the parameter k. The most favorable form of single-path with failover is SP-∞, where *all* neighbors with lower cost will be kept as a backup.

Note the two following remarks about SP-k routing:

- The topology of SP-∞ routes is exactly the same as that of the $SP\text{-}AP$ anypath routes determined with single-path metrics, as described in Section 6.5. So for disconnections and dead-ends the robustness results given for SP-∞ are the same for anypath routing with single-path metrics.

- We pointed out that it is somewhat unfair to compare ease of disconnection of anypath routes with that of single-path routes. We should therefore also point out that the class of multi-path routing algorithms that we represent using SP-∞ are usually *designed* with robustness to link or node failures in mind (see Chapter 2). The first objective of anypath routing is to reduce the cost of forwarding packets. Therefore, we are evaluating anypath based on a criterion that is not part of its design, while comparing it with approaches that explicitly target this criterion.

6.6.3 Route Disconnections and Dead Ends

Since disconnecting an anypath route usually requires more link/node deletions than disconnecting a single-path route, one may be tempted to infer a priori that anypath routes will be less easily disconnected than single path routes. This is in fact not so obvious: while a single link or node failure is enough to disconnect a single-path route, this route touches far fewer links and nodes than an anypath route. Consequently, when a few nodes or links are chosen at random to be deleted, the probability that they are part of a given single-path route is smaller than for the corresponding anypath route between the same source and destination.

To evaluate the robustness of different routing algorithms to disconnection, we ran the following simulation experiment. First, we generate a graph $\mathcal{G}$. From this graph we generate two derived topologies $\mathcal{G}_s$ and $\mathcal{G}_t$ using either of the link or node addition/deletion models, with the same variation parameter p_v. We compute the shortest-(any)path routes from every node to the destination in $\mathcal{G}_s$, and then project each shortest path route from $\mathcal{G}_s$ onto $\mathcal{G}_t$ and verify if it remains connected after the projection.

In so doing, we count the number N_d of *route disconnects* going from $\mathcal{G}_s$ to $\mathcal{G}_t$. Noting $\mathcal{R}_t(i)$ as the route from node i to the destination, and defining $conn(\cdot)$ as the function returning 1 if a route is connected and 0 otherwise, we can express N_d as follows:

$$N_d = \frac{1}{|\mathcal{G}|} \sum_{i \in G} conn(\mathbf{proj}(\mathcal{R}_t(i), \mathcal{G}_s)),$$

where $\mathcal{G}$ is the base graph from which $\mathcal{G}_t$ and $\mathcal{G}_s$ are derived.

We repeated the above experiment 10000 times for each particular choice of time-varying network model, its variation parameter p_v, and routing metric. Figure 6.15 shows the empirical cumulative density function (CDF) of N_d with asynchronous duty cycling, over a network of 100 nodes. Anypath routing has significantly less disconnections than single-path in either the link or node addition/deletion models. With both models, anypath has less than 10 disconnected nodes with empirical probability over 0.85; whereas single-path has less than 10 disconnected nodes with probability only 0.22 and 0.07. These plots also show the number of disconnections for single-hop with failover, in the most favorable configuration (SP-∞, no bound on number of backup routes). With failover, the number of disconnections significantly decreases in comparison with SP-0, but it still remains above anypath. Another observation is that the difference between ERS-best and ERS-any is small; this is coherent with the fact that with asynchronous duty-cycling, average out-degrees are comparable between both of these ERS policies (Figure 6.11).

In Figure 6.16, we show plots for the same set up as Figure 6.15, the only difference being that the network now contains 1000 nodes. Unsurprisingly, with SP-∞ and SP-0 the probability of having less than 10% of routes disconnected is higher than in Figure 6.15: routes are longer, and therefore more likely to break. What is more surprising is that the probability *is lower* for anypath routing. This indicates that anypath routes remain robust at scale. Finally, Figure 6.17 shows the CDF of the number of routes that contain at least one dead end. Anypath routes are more robust under this criterion as well, due to their using more candidate relays at each node.

The number of possible combinations of network size, time-varying model, parameter p_v, and link metric is too large to present CDF plots for each one. We show in Table 6.1 a summary of the robustness to disconnections of anypath and single-path routing under various conditions. Each percentage in the table is the empirical probability of having over $\frac{n}{10}$ disconnected routes in a network of size n. These numbers confirm that the robustness of anypath routes holds for a wide range of configurations.

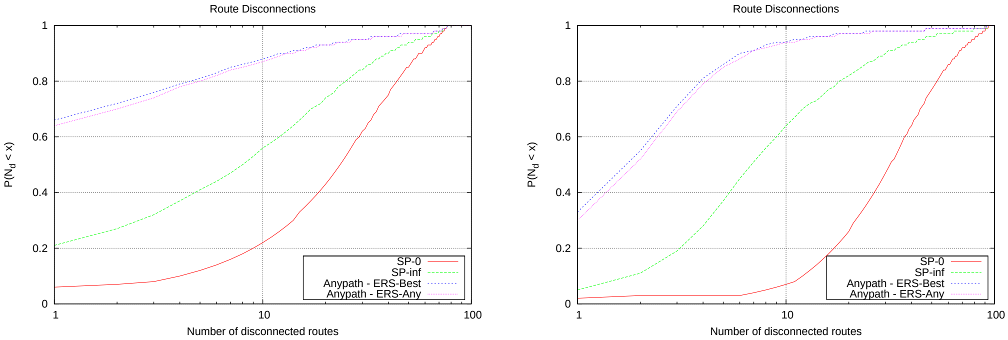
Transmission-Count

		Cut	SP-0	SP-∞	Anypath (ERS-any)	Anypath (ERS-best)
$n = 100$	$p_v = .05$	**L**inks	93%	38%	14%	6%
		Nodes	52%	22%	9%	4%
	$p_v = 0.1$	L	95%	71%	47%	23%
		N	78%	44%	23%	12%
$n = 1000$	$p_v = .05$	L	100%	100%	100%	98%
		N	100%	98%	72%	36%
	$p_v = 0.1$	L	100%	100%	100%	100%
		N	100%	100%	97%	74%

Asynchronous duty-cycling

		Cut	SP-0	SP-∞	Anypath (ERS-any)	Anypath (ERS-best)
$n = 100$	$p_v = .05$	**L**inks	93%	38%	6%	6%
		Nodes	52%	22%	4%	4%
	$p_v = 0.1$	L	95%	72%	27%	25%
		N	78%	44%	13%	12%
$n = 1000$	$p_v = .05$	L	100%	99%	6%	5%
		N	100%	90%	3%	3%
	$p_v = 0.1$	L	99%	99%	19%	16%
		N	100%	99%	9%	8%

Table 6.1: Empirical probability of having over $\frac{n}{10}$ nodes disconnected, for network sizes $n = 100$ and $n = 1000$, and for different parameters p in either the node or link addition/deletions models. Top: Asynchronous duty cycling with energy-cost metric. Bottom: Always-on network with ARQ, transmission-count metric.



(a) Node additions/deletions
$p_v = 0.1$

(b) Link additions/deletions
$p_v = 0.05$

Figure 6.15: Empirical CDF of N_d, the number of disconnected routes, for networks with 100 nodes. The network model is asynchronous duty cycling with transmission energy cost metric.

6.6.4 Route Dilation

To evaluate how much dilation results from imperfect knowledge for different routing algorithms, we ran the following simulation experiments. We generate a graphs $\mathcal{G}$ and two derived topologies $\mathcal{G}_s$ and $\mathcal{G}_t$ as in the previous experiments. For each source i, we compute the shortest-(any)path route $\mathcal{R}_t(i)$ to the destination in $\mathcal{G}_s$. We then project this route onto $\mathcal{G}_t$ and compute its cost. We also compute the shortest path route $\mathcal{R}_s(i)$ in $\mathcal{G}_t$ and compute its cost. We can then compare the difference in cost between both routes. The quantity of interest is then:

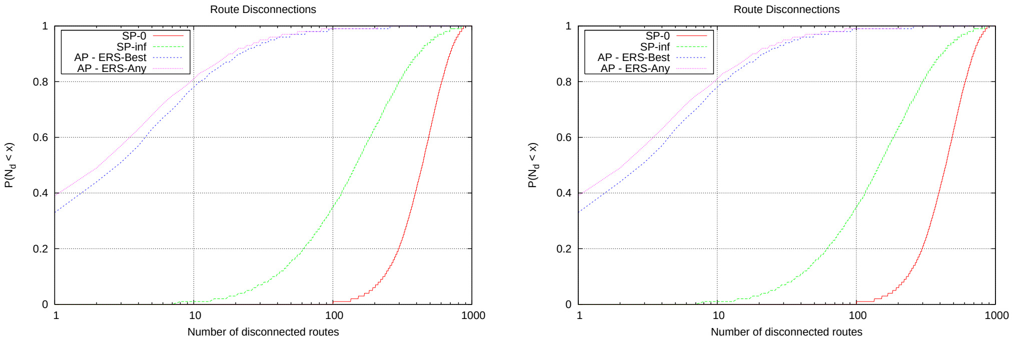
$$d(i) = \frac{Cost(\mathbf{proj}(\mathcal{R}_t(i), \mathcal{G}_s))}{Cost(\mathcal{R}_s(i))}.$$

We call $d(i)$ the *dilation* of the route from i to the destination when the network changes from $\mathcal{G}_t$ to $\mathcal{G}_s$. The route dilation is only defined if $\mathbf{proj}(\mathcal{R}_t(i), \mathcal{G}_s)$ is connected and if a route $\mathcal{R}_s(i)$ actually exists.

We compute the *average* dilation over all nodes in the network:

$$d = \frac{1}{|\mathcal{G}|} \sum_{i \in G} d(i),$$

and then repeat the entire experiment 1000 times with different realizations of $\mathcal{G}_s$ and $\mathcal{G}_t$ for each particular choice of time-varying network model, its variation parameter p_v, and routing metric.



(a) Node additions/deletions
$p_v = 0.1$

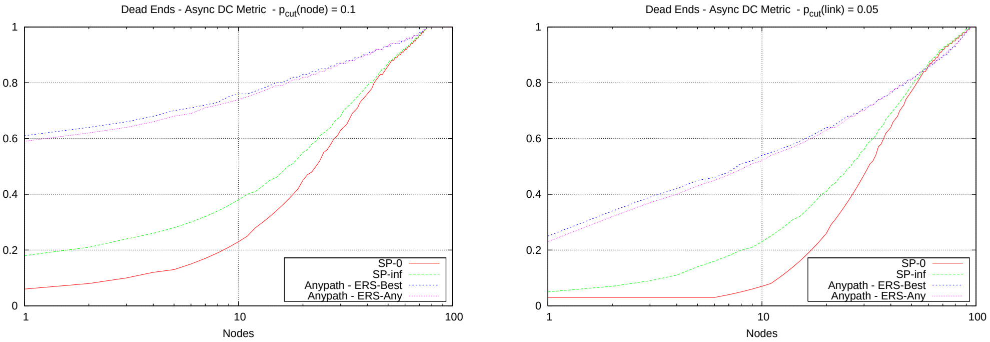
(b) Link additions/deletions
$p_v = 0.05$

Figure 6.16: Empirical CDF of N_d, the number of disconnected routes, for networks with 1000 nodes. The network model is asynchronous duty cycling with transmission energy cost metric.

We show the results of this experiment in Figure 6.18. Here we used unreliable ARQ links with the ETX cost metric and the link fluctuation model. The inflation of anypath routes is always lower than single-path. This confirms the intuition that anypath should be more stable to link fluctuations, because the cost of an anypath route is averaged out over a larger number of links than single-path route. If the noise in the estimation of link delivery probabilities has zero mean, then the cumulative effect should be less strong than for single-path routing.

6.7 Summary

In this chapter, we evaluated using numerical simulation the performance of shortest anypath routing. We introduced two new effective relay selection policies: ERS-all and ERS-any. The resulting routes are more costly with ERS-all and ERS-any than with ERS-best, however they allow the design of relay arbitration protocols with sharply lower overhead. We showed how the ERS policy is *factored into* the shortest anypath routes, and in particular the striking form that it gives to routes when using ERS-all. Finally, we evaluated the robustness and stability of anypath routing in the face of fluctuations and imperfect protocol information.



(a) Node additions/deletions
$p_v = 0.1$

(b) Link additions/deletions
$p_v = 0.05$

Figure 6.17: Empirical CDF of the number of routes with dead ends, for networks with 100 nodes. The network model is asynchronous duty cycling with transmission energy cost metric.

6.8 Appendix: Computation of Remaining Path Costs

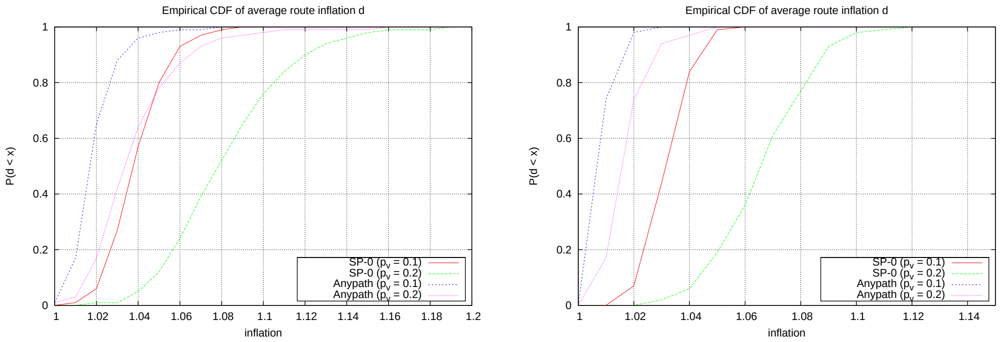
In this appendix we compute the expressions of RPC for ERS-any and ERS-all in the special case where all nodes in $C(i)$ have the same probability of receiving a packet from i. The resulting expressions simplify considerably compared to those for the general case, and are of practical help (both in simulation and in real implementation) to reduce the overhead of computing the RPC for each possible candidate relay set in the Bellman equation (5.2).

ERS-any

We first note that the probability that the outcome of the forwarding phase is $\mathbf{A}$ depends only on the number of components $\mathbf{A}_k$ that are non-zero. With equal reception probabilities, we can note $q_{ij} = q$ to lighten notation, and so:

$$\begin{aligned} P(S) &= \prod_{j \in C(i)} \left(q \mathbf{1}_{j \in S} + (1-q) \mathbf{1}_{j \notin S} \right) \\ &= q^{|S|} (1-q)^{|C(i)|-|S|} \\ &\overset{(a)}{=} P(|S|), \end{aligned}$$

where we emphasize in (a) that in this special case $P(S)$ depends only on the size of the set S. Noting $n = |C(i)|$, we can now rewrite (6.2) as:



(a) $n = 100$ (b) $n = 1000$

Figure 6.18: Empirical CDF of d, the average route inflation, for networks with 100 and 1000 nodes. The link model is unreliable ARQ links with ETX metric. The network changes according to the link fluctuation model, with $p_v = 0.1$ and $p_v = 0.2$. Links in the base graph $\mathcal{G}$ have delivery probability $p = 0.5$.

$$\begin{aligned}
\mathrm{E}\left[f(\mathbf{A})\right] &= \sum_{S \in 2^{C(i)}} P(S) f(S) \\
&= \sum_{\substack{S \in 2^{C(i)} \\ |S|=1}} P(S) f(S) + \sum_{\substack{S \in 2^{C(i)} \\ |S|=2}} P(S) f(S) + \ldots + \sum_{\substack{S \in 2^{C(i)} \\ |S|=n}} P(S) f(S) \\
&\overset{(a)}{=} P(1) \sum_{\substack{S \in 2^{C(i)} \\ |S|=1}} f(S) + P(2) \sum_{\substack{S \in 2^{C(i)} \\ |S|=2}} f(S) + \ldots + P(n) \sum_{\substack{S \in 2^{C(i)} \\ |S|=n}} f(S) \\
&= \sum_{k=1}^{n} \left(P(k) \sum_{\substack{S \in 2^{C(i)} \\ |S|=k}} f(S) \right) \\
&\overset{(b)}{=} \frac{1}{n} f(C(i)) \sum_{k=1}^{n} P(k),
\end{aligned} \tag{6.4}$$

where we used in (a) that $P(S)$ does not depend on the individual nodes in S, and in (b) the fact that each node of $C(i)$ is present in an equal number of subsets S, and f selects a node uniformly at random from S. Finally, note that the union of events $|\mathbf{A}| = k$ for $k = 0, \ldots, n$ form a partition of all possible outcomes. The sum over $P(k)$ (b) is over $k = 1, \ldots, n$ and is therefore equal to $1 - P(0)$, or the probability that at least one node in $C(i)$ receives a packet.

To obtain the expression of the RPC $R_{iC(i)}$, we substitute (6.4) into (6.1), and find:

$$R_{iC(i)} = \frac{1}{n}\sum_{j=1}^{n} D_j.$$

ERS-all

As for ERS-any, we split the $\mathrm{E}\left[f(\mathbf{A})\right]$ into separate sums for each value of $|S|$:

$$\mathrm{E}\left[f(\mathbf{A})\right] = \sum_{k=1}^{n}\Bigg(P(k)\sum_{\substack{S\in 2^{C(i)}\\ |S|=k}} f(S)\Bigg). \tag{6.5}$$

At this point however we cannot bring $f(S)$ out of the inner sum, because with ERS-all $f(S)$ does depend on S and not only just on $|S|$. Fortunately, we can still avoid the combinatorial enumeration of subsets of $C(i)$. Noting that the inner sum in the above line is over $\binom{n}{k}$ possible subsets S, that there are k nodes in each subset, and that each node appears an equal number of times in the $\binom{n}{k}$ combinations, we have that

$$\sum_{\substack{S\in 2^{C(i)}\\ |S|=k}} f(S) = f\big(C(i)\big)\binom{n}{k}\frac{k}{n}. \tag{6.6}$$

Substituting (6.6) into (6.5) and the result into (6.1), we obtain the expression of the remaining path cost with ERS-best for the case where each node in $C(i)$ has equal probability of receiving the packet:

$$\begin{aligned}
R_{iC(i)} &= \frac{1}{1-(1-q)^n} f\big(C(i)\big)\sum_{k=1}^{n} P(k)\binom{n}{k}\frac{k}{n}\\
&= \frac{1}{1-(1-q)^n} f\big(C(i)\big)\sum_{k=1}^{n} q^k(1-q)^{n-k}\binom{n}{k}\frac{k}{n}\\
&= \frac{1}{1-(1-q)^n} f\big(C(i)\big)\sum_{k=1}^{n} q^k(1-q)^{n-k}\frac{n!}{k!(n-k)!}\frac{k}{n}\\
&= \frac{1}{1-(1-q)^n} f\big(C(i)\big)(n-1)!\sum_{k=1}^{n} q^k(1-q)^{n-k}\frac{1}{(k-1)!(n-k)!}.
\end{aligned} \tag{6.7}$$

Chapter 7

Protocol Design and Prototype Implementation

7.1 Overview

Our development of anycast forwarding and anypath routing in Chapters 4 and 5 assumed an intentionally simple network model. In Chapter 6, we then gained some further insight into some protocol design tradeoffs through a simulation-based study.

After this, the logical next step in the validation of anypath routing is by means of a practical study. In this chapter, we describe the design and implementation of an anypath routing protocol that aims to increase energy efficiency for data collection in *low-rate, low-power* multi-hop wireless networks. The implementation was done using the TinyOS [64] operating system and was evaluated on a 50-node testbed of *TinyNode* [23] embedded wireless devices. These are essentially made of a simple microcontroller and a low-rate (up to 150 kilobits/sec) radio; as part of this work we have ported two generations of TinyOS to this platform and developed a complete software driver for the radio transceiver. By evaluating the protocol on a wireless testbed, we seek to confront anypath routing with some of the vagaries of real low-power wireless networks, namely:

- **Wireless channels:** As we have previously discussed, wireless channels vary over time, including with stationary nodes [13, 32, 114]. Any routing protocol must cope with imperfect inputs and be robust to link fluctuations.

- **Resource constraints:** Nodes have 10 Kilobytes of RAM and 48 Kilobytes of ROM. In such an environment, each byte counts, and system components must be very carefully designed so as to keep the overall memory footprint as low as possible.

- **Computational constraints:** Nodes use a simple microcontroller running at 8Mhz. Any non-trivial computations are prohibited; even simple operations such as in-memory copying of packets should not be done carelessly at multiple points in a protocol stack, if they can be avoided.

Key protocol aspects

The two key aspects of our protocol and implementation, tentatively named Low-power Anypath Routing Protocol (LARP) are:

- **Hybrid anycast forwarding:** LARP combines aspects of both asynchronous duty-cycling and individual synchronous duty-cycling, in a way that is inspired by WiseMAC but generalized to anycast forwarding. Nodes advertise their wakeup schedules to their neighbors, using periodic broadcasts or piggybacking on data packets. A node with a packet to send first traverses its forwarding table, that contains schedule and distance information associated with each candidate relay. If schedule information for every candidate is absent or stale[1], then the sender uses the asynchronous duty cycling forwarding mechanism of Section 4.5, allowing reduced-length preambles over unicast forwarding. If the sender has schedule estimates of different precision for each candidate relay, then it uses the candidate whose schedule is most precisely known, in order to minimize the required preamble length. Finally, if the schedule information for all candidates is precise, our node selects the next available relay, thus reducing delay in comparison with unicast forwarding.

- **Anycast delivery:** In many multi-hop wireless networks, and in particular in wireless sensor networks, the dominant traffic pattern is each node sending data to the sink; this sink serves as a gateway to the wired network. In practice, networks frequently use more than one sink, in order to increase redundancy and reduce the average distance from each node to the closest sink. The LARP protocol supports an anycast service, where a packet is sent to any sink, without preference or a priori choice by the sender. The underlying anycast forwarding fits very well with a network-layer anycast service, and we show how to adapt anypath routing to this type of service model.

---

[1] Recall that clocks drift, such that after several minutes without synchronization a long preamble is again necessary as in asynchronous duty cycling.

7.2 Protocol Design

In this section we delve into more detail into the design of LARP. One key to the protocol design proposed here is that it *merges* different strategies that were seen individually in previous Chapters. In particular, we use a hybrid form of anycast forwarding that combines both aspects of individual synchronous duty cycling and of asynchronous duty cycling. Also, LARP combines both ERS-all and ERS-any, using each one where it is most appropriate.

7.2.1 Hybrid Anycast Forwarding

The development of anycast forwarding for asynchronous duty cycled link layers, in Chapter 4, showed how it is possible to reduce the length of preambles by a factor of 2 to 5 for realistic sizes of candidate relay set. Yet, even with such a reduction, the resulting preamble lengths are still longer than the packets themselves. This is the price to pay for the simplicity and robustness of having no synchronization, and thus avoiding the messaging overhead of establishing and exchanging schedules. When is such an approach worthwhile? The answer depends of course on the relative rates of data traffic and control traffic required to maintain synchronization: if data traffic is extremely sparse, then the cost of synchronization control traffic can become higher than that of forwarding data packets; at some point in this tradeoff it becomes overkill to maintain schedules and the asynchronous form is more efficient.

Motivated by this tradeoff, LARP uses a hybrid anycast forwarding combining aspects of both asynchronous duty-cycling and individual synchronous duty-cycling. This link-layer scheme is reminiscent of WiseMAC [25], but transposed to anycast forwarding.

This hybrid anycast forwarding link layer can operate as the asynchronous duty cycling scheme of Section 4.5, as the individual synchronous duty cycling scheme of Section 4.6, or in an intermediate regime between these two boundary points. The operating regime is driven by the current information that a node has about its neighbor's schedules. If a node has no schedule information for its neighbors, then anycast forwarding is done as in asynchronous duty cycling, with repeated transmissions until a node receives the packet. At the other extreme, if a node has exact schedule information for all of its neighbors, then anycast forwarding is sender-driven, with the sender selecting the node that is next to awaken.

In the general case, the sender may have schedule information for a subset of its neighbors. Also, schedule information "ages", due to the fact that node clocks drift. So a schedule for the next wake-up time of a neighbor i is not simply represented by a value Δ_i, but rather by an interval $[\Delta_i - \frac{\sigma_i}{2}, \Delta_i + \frac{\sigma_i}{2}]$. We call this the *drift* *window.* The value σ_i represents the uncertainty, or imprecision, associated with a node's next wakeup time, and it increases linearly with time elapsed since the last synchronization point. Our design of LARP makes the assumption that the effective wakeup time of a node is uniformly distributed within the drift window. Under this assumption, a node wishing to send a packet to this neighbor i must precede it with a preamble of length σ, and start transmission at time $\Delta_i - \frac{\sigma_i}{2}$, in order to have a preamble that covers the temporal window within which i is expected to awaken.

Note that our assumption of uniformly distributed wakeup time within the drift window is a simplifying assumption. In reality relative clock drifts between two nodes are not uniformly distributed; if one node's clock runs faster than another then its effective wakeup time will be biased towards the "early" part of the above interval. (Relative clock drifts are not constant however; changing temperatures in particular affect the drift of crystal oscillators). Note that the use of hybrid anycast forwarding does not preclude more complex schemes that attempt to measure and track clock drifting [30]; we do not however investigate these here.

In the general case, each neighbor's last schedule update was received at a different time, and so schedules have varying degrees of uncertainty. In other words, neighbor schedules will have different values of σ. Different strategies for anycast forwarding are possible when schedule precisions are non-homogeneous. Assume that we have n candidate relays, each with an associated (Δ_i, σ_i). Note that as in Section 4.5, all nodes have the same wakeup interval t_{rx}; it is therefore possible to represent an unknown schedule as $[-\frac{t_{rx}}{2}, \frac{t_{rx}}{2}]$. Some possible strategies to choose amongst multiple candidate relays with differing (Δ_i, σ_i) are:

- **Energy minimization:** The sender chooses the node i with lowest σ_i.
- **Delay minimization:** The sender chooses the node i with lowest Δ_i.
- **Joint minimization:** One can also design criterion that minimize an objective function incorporating both Δ_i and σ_i. For example, one might select the node with lowest sum $c_1\Delta_i + c_2\sigma_i$, with c_1 and c_2 being the relative weights given to energy and delay.

LARP seeks to minimize transmission energy first and foremost. Consequently, it uses the following strategy: consider a sender i with candidate relay set $C(i)$. First, the sender searches for the candidate in $C(i)$ with smallest σ_i. Call this candidate j. Then, the sender computes[2] the transmission-cost ALC d_{iJ}^{async} using the asynchronous anycasting of Section 4.5, and determined by equation (4.13) (p. 79). The quantity

---

[2] Or uses a lookup table, as in our implementation.

$\sigma_j + t_{pkt}$ represents the transmission duration to send the packet to the candidate relay whose schedule is most precisely known, and the quantity d_{iJ}^{async} represents the expected transmission duration to send the packet with receiver-driven anycasting. Our sender can therefore compare both quantities, and use the forwarding strategy that is the least costly. This strategy is therefore a *hybrid* one, that combines both aspects of asynchronous and synchronous duty cycling, and uses each one to its best advantage.

Anycast Link Cost

Having described the hybrid anycast forwarding scheme used by LARP, we must now design a metric for the anycast link cost (ALC) to transmit a message to from i to any node in J. We call this link cost d_{iJ}^{hyb}, we can note first that

$$d_{iJ}^{hyb} \leq d_{iJ}^{async},$$

where d_{iJ}^{async} is the anycast forwarding cost using asynchronous duty cycling (equation (4.13), p. 79). This inequality follows directly from the strategy described in the previous section: the sender either uses asynchronous anycast forwarding, *or* uses the sender-driven (synchronous) scheme if that one has lower expected cost.

Finding a closed-form solution for d_{iJ}^{hyb} is however more complex than establishing this inequality, and would require making specific assumptions on the rate at which schedules are refreshed, as well as the traffic patterns. We use instead the expression

$$d_{iJ}^{hyb} = t_{pkt} + \frac{t_{rx}}{|C(i)| + 1},$$

which is the same as the latency ALC of Definition 4.13 (p. 84). Our choice of this ALC over, for example, the d_{iJ}^{async} is motivated by two simple reasons. The first is the inequality defined above, and the fact that we prefer to *under*-estimate than over-estimate the ALC, so as err on the side of too many rather than too few candidate relays. The second is that the asynchronous duty cycling ALC is less stable to imprecise estimates of candidate relay set size (Figure 4.9, p. 80).

7.2.2 Network-Layer Anycast

While the idea of anycasting at the link-layer is relatively recent one, the anycasting service model, where a message is delivered to any one of a set of nodes, is itself not new. Network-layer anycast routing has been proposed and used for many years in the Internet [81]. It is useful in general as a mechanism to provide redundancy and load-balancing. Three examples of the use of anycast routing in the Internet are for advertising IPv6 to IPv4 translation gateways [41]; sending a domain name (DNS) lookup request to any name server for a given root when multiple identical servers are operated; and to create multiple instances of Rendez-vous Points for protocol independent multicast sparse-mode (PIM-SM) [54]. In the Internet, anycast routing is usually implemented by having routers at different places in the network simultaneously advertise the same destination IP address range. This results in anycast-addressed packets being routed to the nearest router advertising the address.

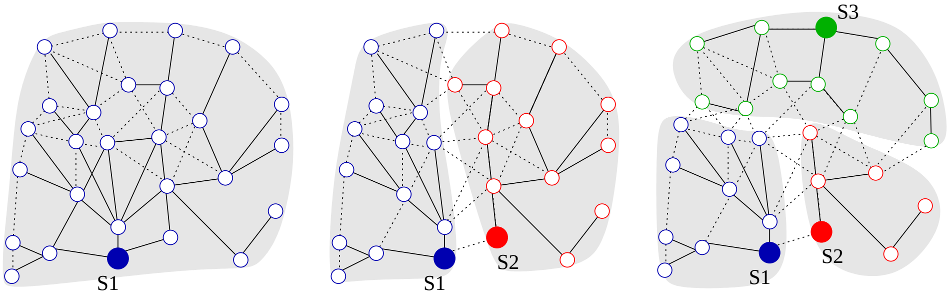


Figure 7.1: Single-path anycast data collection with $1, 2$, and 3 sinks. The underlying topology is shown with dotted lines, and route from each node to its nearest sink is shown with solid lines.

In the context of a wireless multi-hop network, anycast routing allows to take advantage of multiple sink nodes by shortening average path lengths and thus reducing relaying loads, energy consumption, and end-to-end packet loss. We illustrate anycast data collection in Figure 7.1.

How does anycasting to one of multiple sinks transpose from single-path routing to anypath? Just as unicast anypath routing generalizes unicast single-path routing, anycast anypath routing generalizes anycast single-path routing. In fact the mechanism to support anycast routing with LARP is exactly the same as with single-path, with sink nodes advertising reachability to the same address.

Intuitively, one might expect that the anycast delivery service model should be a “good match” for an anypath routing protocol, since anypath routing itself is predicated upon an underlying link-layer anycast primitive. And indeed, the relative reduction in routing cost when anycasting to multiple sinks is for certain nodes greater when the underlying routing is anypath than with single-path. To see this, consider the following: with single-path routing, network-layer anycasting results in a packet being delivered to the *closest* node in the anycast group. Thus, the anycast routing distance is equal to the minimum distance from a node to each of the anycast destinations. With anycast forwarding however, this is true only for *some* nodes. Specifically, nodes that are at equal distance to multiple anycast destinations will have a larger candidate relay set, and thus their cost to reach an anycast destination will be lower than the cost to reach the closest destination alone. This is illustrated in Figure 7.2.

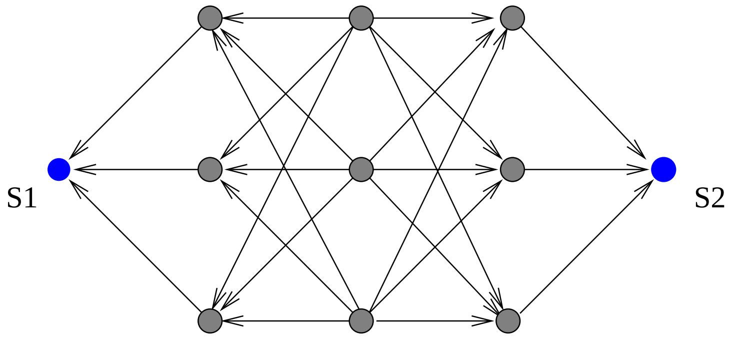


Figure 7.2: Network-layer anycasting with anypath routing. The two blue nodes are sinks; packets from intermediate nodes can be delivered to any sink. Each of the three nodes in the middle vertical row has 6 candidate relays. With only one sink, each node in the middle row has only 3 candidate relays. The cost to deliver a packet from these middle nodes is thus *lower* than with any of the two sinks alone. In contrast, with single-path routing, the cost to deliver a packet is always *equal* to the minimum cost using one of the sinks alone. The relative gains from having multiple sinks are stronger with anypath routing than with single-path routing.

7.2.3 Effective Relay Selection and Arbitration

With hybrid anycast forwarding, many packet transmissions are *sender-driven*, when the sender has a precise enough schedule for a candidate relay that it chooses to transmit directly to that node. However, when schedule information is too loose, anycast forwarding becomes receiver-driven, and we have seen in Chapters 5 and 6 that the choice of Effective Relay Selection (ERS) strategy plays has important effect on performance with receiver-driven anycast forwarding. A contribution of Chapter 6 was to investigate the use of different effective relay selection (ERS) policies, and the design of LARP is guided by two observations from that chapter. The first observation is that with asynchronous duty cycling, ERS-any and ERS-best have near-identical performance. The second observation is that at short distances from the destination, ERS-all can be used with little additional cost compared to ERS-any.

An important objective is for LARP to require the simplest possible relay arbitration protocol, given that we are targeting low traffic rates, and thus cannot amortize arbitration over large numbers of packets (as for example in ExOR). With this objective in mind, LARP uses a combination of ERS-all and ERS-any. This choice is done on a *per-packet* basis, according to the following procedure: when a packet is passed down to the link-layer of a node i for anycasting to any neighbor in set J, the link layer estimates the remaining path costs R_{iJ}^{any} and R_{iJ}^{all} corresponding to either ERS policy. If the difference is large, i.e., if $R_{iJ}^{any} < R_{iJ}^{all} + K_{ers}$, where K_{ers} is a protocol constant, then the packet is sent using ERS-any. Otherwise, the packet is sent using ERS-all. This choice is indicated in the packet header; our current implementation uses two different anycast addresses, one corresponding to ERS-all relaying, and one to ERS-any. One feature that is notably missing from our current system is the design and implementation of a relay arbitration protocol for cases when a packet is sent using ERS-any. This will be necessary whenever paths are several hops long, and when large intervals elapse between node schedule exchanges.

7.3 XE1205 Radio and TinyNode Platform

We implemented LARP on the TinyNode wireless platform that is produced by Shockfish SA. The TinyNode is an wireless embedded module targeted at wireless sensor networks. It is roughly comparable to existing platforms such as Telos [86] or EyesIFX [37]; one significant difference is that it has higher range than other low-power wireless platforms, achieving over 500m [23] in non-line of sight urban environments, in comparison with 60-80m for the Telos node. This extended range motivated our choice of this platform for the SensorScope project, an application of wireless sensor networks to environmental monitoring that we started in 2005. Our choice of this particular platform for our LARP implementation is chiefly motivated by the flexibility of its radio transceiver. It can be controlled on a byte-per-byte basis, as opposed to transceivers such as the CC2420 (used in most other designs today) that are controlled on a packet level and with which implementing fine-grained link-layer operations is not possible.

7.3.1 Platform Overview

The design philosophy of TinyNode is to place core components which are required for every application on a small module, and additional functionality on extension boards. The core module, shown in Figure 7.3, is a versatile low-power wireless node, and comes with an array of extension hardware offering a wide set of connectivity, storage, energy, and interfacing options. It uses a low power transceiver which has energy characteristics comparable to those found on other sensor nodes (“motes”), but offers a significantly larger range, and bit rates from 1.2kbps all the way up to 152kbps. The platform comes with full TinyOS support, including a complete radio stack, support for network reprogramming with Deluge, [40] and bridging software for GPRS/GSM data transfer.

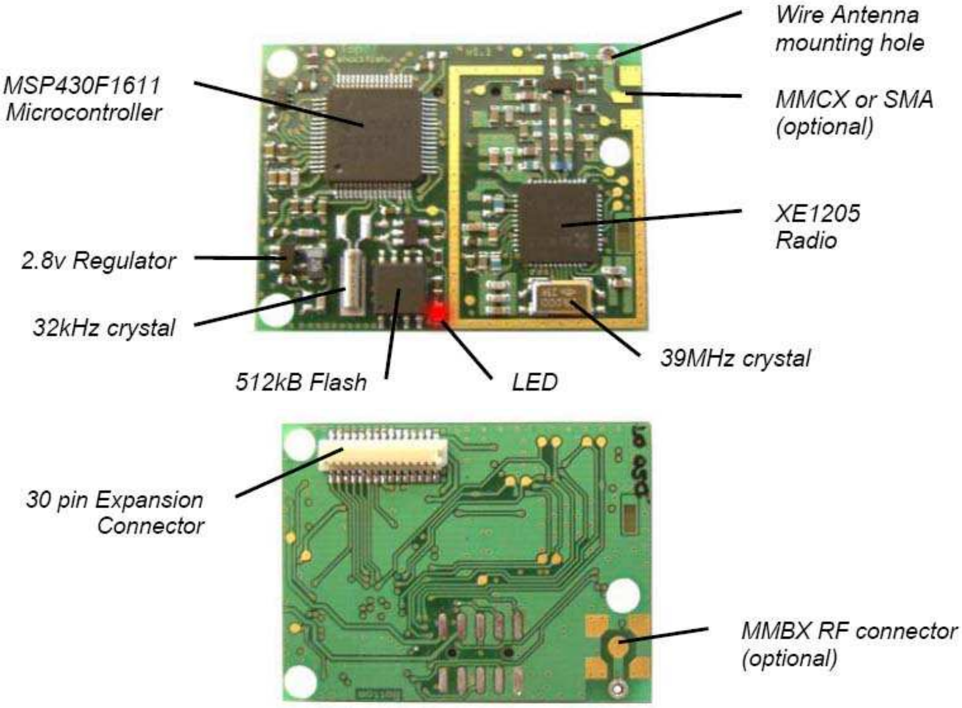

Wire Antenna mounting hole
MSP430F1611 Microcontroller
MMCX or SMA (optional)
XE1205 Radio
2.8v Regulator
32kHz crystal
39MHz crystal
512kB Flash
LED
30 pin Expansion Connector
MMBX RF connector (optional)

Figure 7.3: TinyNode core module (upper and lower sides).

MSP430 Microcontroller

The TinyNode features a MSP430F1611 ultra-low power microcontroller that is fully supported by TinyOS and has the lowest power consumptions and fastest wake-up cycles available today. The digitally controlled oscillator (DCO) allows wake-up from low-power modes to active mode in less than 6μs and may operate up to 8MHz. Typically, the DCO will turn on from sleep mode in 300ns at room temperature. The MSP430F1611 has two built-in 16-bit timers, a fast 12-bit A/D converter, dual 12-bit D/A converters, one or two universal serial synchronous/asynchronous communication interfaces (USART), I2C, DMA, and 48 I/O pins. The same microcontroller is used on the Telos [86] and EyesIFX [37] platforms. We refer to [86] for a full comparison of the MSP430F1611 with competing microcontrollers from Atmel, Motorola, and Microchip. The core module also has a 4Mbit flash chip that can be used for storing several firmware images or for logging data.

XE1205 Radio

Particular attention has been put to the choice of the radio transceiver. The XE1205 from Semtech (formerly XEMICS) is an integrated transceiver that can operate in the 433, 868 and 915MHz license-free ISM frequency bands. The version used for the measurements given in this paper operates at 868Mhz.

All major RF communication parameters are programmable and most of them can be dynamically set. The XE1205 offers both narrow-band and wide-band communication with the same hardware configuration, allowing data rates from 1.2kbit/s to 152kbit/s. Compared to other transceivers in the market (including Chipcon, Nordic, RFM, Micrel, TI, Infineon), the XE1205 offers the highest link budgets available today in the license free ISM bands. With an output power of +15dBm and sensitivity of -116dBm at 4.8kbit/s, a link budget of 131dB can be achieved. This is 22dB better than for the Chipcon CC1000 radio used on the Mica2 platform, which gives the TinyNode approximately 4 times longer range.

Table 7.1 shows the key transceiver characteristics for the CC1000, CC2420, and XE1205 radio transceivers. The link budget is the sum of all signal gains and losses over the entire wireless path, and the receiver sensitivity is the signal level at which the decoded signal has a bit error rate (BER) below 0.1%. For comparison, the antenna gain is assumed to be unitary (0dBi) for all platforms and the outdoor range is calculated according to an isotropic path loss model with a gain exponent of n=2.6 for open field propagation.

Platform	**Mica2**		**Telos Sky**	**TinyNode**	
Transceiver	CC1000		CC2420	XE1205	
Frequency	869 Mhz		2.4 Ghz	869 Mhz	
Max. Tx Power	5dBm		0 dBm	15dBm	
Data Rate	76.8 kbps	4.8 kbps	250 kbps	76.8 kbps	4.8 kbps
Sensitivity	-98 dBm	-104 dBm	-94 dBm	-106 dBm	-116 dBm
Link Budget	103 dB	109 dB	94 dB	121 dB	131 dB
Range Outdoor[3]	160m	300m	80m	600m	1800m

Table 7.1: Comparison of radio transceiver characteristics.

Energy consumption.

Energy consumption is a critical parameter of a sensor node. Table 7.2 shows the current consumption for Mica2, Telos, and TinyNode. The microcontroller related consumptions of the TinyNode, and Telos nodes are identical since they use the same chip. TinyNode has radio consumptions comparable to the Mica2, while offering significantly higher range and data rates. Telos also has comparable radio consumption, but the CC2420 offers a higher bit rate and faster radio wake-up. The tradeoff to this lower consumption is a reduced communication range.

	Mica2	**Telos Sky**	**TinyNode**	
Min Voltage	2.7	1.8	2.4	V
Max Voltage	3.3	3.6	3.6	V
MCU sleep with RTC on (LPM3)	19	5.1	5.1	μA
MCU active	8	1.8	1.8	mA
MCU active, Radio RX	15.1	21.8	15.8	mA
MCU active, Radio TX at 0dBm (1mW)	25.4	19.5	25	mA
MCU active, Flash Read	9.4	4.1	5	mA
MCU active, Flash Write	21.6	15.1	16	mA
MCU wake-up latency	180	6	6	μs
Radio wake-up latency	1800	580	1500	μs

Table 7.2: Current consumption and wake-up times.

	Mica2 1% duty cycle	**TinyNode 1% duty cycle**	**TinyNode 0.2% duty cycle**
Bit Rate	19.2 kbps	152 kbps	152kbps
Listen Time	8 ms	1.9 ms	1.9 ms
Listen Period (Max. Latency)	1085 ms	190 ms	950 ms
Max throughput	0.89 pkts/sec	5.5 pkts/sec	1.05 pkts/sec
Average Power Consumption	509μW	489μW	104 μW
Theoretical lifetime with 2 x AA alkaline cells, 2000mAh	1.3 years	1.4 years	6.6 years

Table 7.3: Power consumption and parameters of our XE1205 asynchronous duty cycling implementation.

7.3.2 XE1205 Radio Device Driver

The port of TinyOS to the TinyNode platform consists essentially of low-level hardware adaptation code and a new radio driver and MAC layer for the XE1205 transceiver. The hardware adaptation phase made full use of the Hardware Abstraction Architecture (HAA) already developed at UC Berkeley and TU Berlin [103] with support for the MSP430 microcontroller.

Unlike the core platform support, the radio stack was written from scratch, since the XE1205 transceiver has not been previously used in TinyOS-supported platform. We have designed and implemented a full radio stack around the XE1205 which includes CSMA, acknowledgement frames, low-power listening, and support for bit rates all the way up to 152kbps. The radio stack is relatively compact (206 bytes of RAM and 6126 bytes ROM, including HPL and BusArbitration code).

The XE1205 interfaces to the microcontroller using an SPI bus. It offers a bytewise read/write interface for sending and receiving data, and is configured with register operations over SPI. Support for full-speed operation at 152kbps would be difficult with a bare bytewise interface (such as that of the Chipcon CC1000), since every single byte (transmitted or received) must be handled in less than 50 μs. As a comparison, the TinyOS driver for the Mica2's CC1000 transceiver operates at 19.2 kbps, giving it up to $416\mu s$ to handle each byte.

Fortunately, the XE1205 includes some functionality which helps to offload the microcontroller. In particular, it offers a 16-byte FIFO buffer for sending and receiving packets, and a hardware preamble detector which generates an interrupt as soon as a configurable preamble (of length 8 to 32 bits) is received. While the FIFO buffer avoids having to read (write) every byte as it arrives (transmits), a latency below 50 μs is still necessary during packet reception of long packets each time the FIFO reaches 16 bytes, and during transmission each time the FIFO becomes empty. At this point, the driver must respond rapidly enough to read (write) 16 bytes from (into) the FIFO, otherwise an incoming byte will be lost, or the outgoing bitstream will contain a gap and lose synchronization.

To evaluate software overhead, we measured processing and switching times by sending a continuous packet stream, with initial backoffs disabled, and computing the total channel utilization. While this is not a realistic application profile, it allows us to evaluate if there is any inefficiency in the packet-processing and switching times. The measurements show that the driver is fast: total channel utilization when sending a continuous packet stream, is 68.8% at 152kbps, 80.2% at 76kbps and 94.7% at 19.2kbps. In comparison, the Mica2 stack running at 19.2 kbps has approximately 85% channel utilization in the same conditions. For TinyNode, utilization decreases with bit rate, because the per-packet overhead has a constant component which is independent of bit rate.

Implementation of Unicast Asynchronous Duty Cycling in TinyOS-1.x

Our TinyOS-1.x link-layer for the XE1205 radio stack implemented asynchronous duty cycling as described in Chapter 2 (Section 2.4). While we rewrote the TinyOS-2.0 driver from scratch, it bears some resemblance to the 1.x driver, in particular concerning the lowest-level interactions with the XE1205 chip, and also in some aspects of the asynchronous duty-cycling parameters. The implementation takes advantage of the

XE1205 pattern detector: when a node wakes up, it programs the pattern detector with a two-byte pattern 1010101.., corresponding to the preamble sequence. If these bytes are received, the radio signals an interrupt, and the node now knows that (with high probability) a packet preamble is ongoing. It then reprograms the preamble detector with a 3-byte start-of-frame sequence, and awaits a second interrupt signalling the start of packet reception. This use of the preamble detector allows to significantly off-load the microcontroller in comparison to a software implementation which must process every byte of the preamble, as with the CC1000.

For real performance, it is interesting to examine a typical asynchronous duty cycling mode with 1% duty cycle, meaning that the receiver is active during 1% of the time for listening. At 152kbit/s, we obtain a minimum listen period of 1.9ms (including radio start-up time and RSSI measurement), which means the listening period is 190ms for a 1% activity. In comparison, due to higher start-up times and lower data rates, a Mica2 node at 19.2kbit/s has 8ms of listening time and a listening interval of approximately 1 second. In comparison, this represents a fivefold improvement in latency (or equivalently, throughput) over Mica2, with comparable battery lifetime and range. Conversely, if an application can tolerate a 1-second per-hop latency but requires minimizing energy consumption, the TinyNode can run at 0.2% duty cycle and consume almost an order of magnitude less than Mica2, whilst offering the same delay as the Mica2 at 1% duty cycle. In this case, a theoretical lifetime of over 6 years can be achieved with 2 x AA alkaline cells. Note that the relative improvement is smaller than the ratio of bit rates, because both listen times include a radio wake-up time. While the XE1205 wake-up time is shorter than the CC1000 wakeup time, its relative duration when counted in byte times at 152kbps is higher than for the CC1000. Table 7.3 summarizes these numbers. Note that due to self-discharge and degradation, another type of battery technology needs to be used to obtain such lifetimes (such as Lithium Thionyl Chloride).

7.4 Architecture and Implementation

This section describes the architecture and implementation of LARP on the TinyNode platform. Unlike the XE1205 driver described previously, our implementation of LARP is a prototype one, and as such we expect it to evolve as we gain further experience with long-running deployments. However, we believe that the architecture and salient features that we describe here will perdure.

7.4.1 Scheduled Link Layer

The XE1205 radio driver manages the physical layer and presents a flexible software interface to the XE1205 transceiver chip. We now move one layer up, and describe the scheduled link layer implementation that runs on top of the bare XE1205 driver. Our current implementation runs with the XE1205 radio set at 76170 bps. The constants given below are based on this underlying bit rate; changing these is of course possible if the radio rate is configured differently.

The scheduled link layer has three primary functions: to set a local schedule and advertise it to neighbors, to populate and maintain a *neighbor table* that contains estimates of link quality to other neighbors, and to track other nodes' schedules.

Our link-layer design is inspired by Polastre et al's [85] recent proposal for a unifying low-power wireless link abstraction. While the programming interfaces of this abstraction are not yet set in stone, certain high-level design decisions are likely to remain. In particular, the choice of maintaining a neighbor table and link estimates at the link layer rather than at the network layer has shown to be useful in isolating low-level hardware characteristics from upper layers, and we expect this design to live on in future systems. We expect that following (in spirit at least) the key design decisions of this link abstraction will facilitate the port of our protocol to other low-power, low-rate wireless platforms.

Advertising

At least two types of information must be exchanged for the purpose of scheduling and link estimation. The first concerns schedules: each node must inform neighbors of its wakeup schedule. The second is related to link estimation: in order for two nodes to estimate the reliability of the link between them, they must (at least) have some idea about the number of packets each one has previously transmitted. For example, a node i receiving a packet from neighbor j for the first time in one hour does not know, without further information, if j has sent several other packets to i which were lost, or if this is indeed the only transmission.

Both types of information are transmitted as part of link-layer framing that is not seen by upper layers. Schedules are encoded in 16 bits; the 16-bit schedule value is filled in by the link layer immediately prior to transmission and represents the interval until the next wakeup period of that node. The granularity of this 16-bit value is in $\frac{1}{8192}$-ths of a second, giving a granularity just over the transmission duration of one byte. Nodes can therefore indicate schedule wakeup times up to 8 seconds ahead; given a listen time of 2 milliseconds (time to turn on and settle the radio, and listen

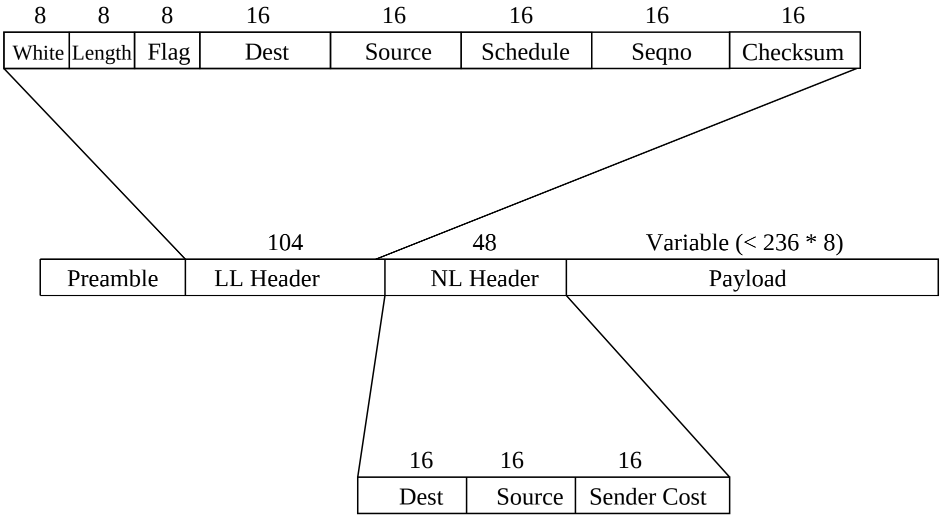


Figure 7.4: Link and network layer framing. Lengths are indicated in bits. The link-layer header contains a whitening byte (XOR'd with the remainder of the packet to prevent long streams of 1's or 0's, that can cause loss of bit-synchronization at the receiver), length, source, and destination fields, as well as schedule and sequence number fields in support of scheduling and link estimation. The "flag" field encodes the type of ERS to be used for this packet, and whether this frame is followed by a network-layer header and payload. Link-layer headers can also be transmitted alone, in which case they may be extended to contain sequence numbers for multiple neighbors.

for a duration equivalent to 8 bytes' transmission time[4]) this allows to support duty cycles below 0.1%.

The link layer frames also contain 16-bit sequence numbers that are used by neighbors for link estimation. These sequence numbers are *per-neighbor*; each node maintains one sequence number per neighbor; broadcast packets are similarly numbered in their own sequence number space. The packet format is shown in Figure 7.4. Link layer frames can also be sent alone, without a preceding a full packet, in order to update a node's schedule and link information with its neighbors.

Link Estimation

LARP currently uses a very simple link estimation algorithm that keeps an count of the (estimated) number of lost packets over a window of k previously transmitted packets from each neighboring node (with $k = 32$ in our implementation). The link estimation algorithm also assumes a *minimum* broadcast rate from each node,

---

[4] With a transmission rate of 76kbps.

allowing it to progressively degrade a link estimate when no packets are received from a neighbor for an extended interval. Our link estimator simply characterizes a link as "on" or "off"; we do not seek to estimate the *probability* of packet delivery or other derived metrics such as ETX. As such it can be classified as a link *filtering* approach according to the model of Section 3.3.1. A link is "on" whenever it has delivered at least $k-4$ of the previous k transmitted packets[5]. If no links satisfy this criterion, it is relaxed until at least 5 neighbors are considered reachable.

Note that unlike many existing approaches, this link estimation scheme does not exchange reverse link estimates in order to compute bidirectional link quality. This simplification is motivated by the observation that link asymmetry is usually less pronounced on highly reliable links; since we only attempt to use highly reliable ("on") links, we assume that asymmetry will pose less problems than if we frequently used intermediate and poor links.

Link-layer broadcasts

An individually scheduled link layer effectively transforms the broadcast wireless channel into a number of pairwise unicast channels. As such, link-layer broadcasts become harder to implement, and have a transmission cost that is higher than that of unicast packets. Broadcasts are carried out by transmitting a packet with a long preamble so as to cover the interval t_{rx} and "hit" all neighbors. Note that in certain cases it may be less expensive to individually transmit to each neighbor than to send a preamble of length t_{rx}; we have not explored this possible optimization further.

7.4.2 Anycast Link Abstraction

From a protocol layering and software interfacing viewpoint, the use of link-layer anycasting requires some significant changes to one of the most essential system calls, namely the function that is invoked by the network layer to transmit a packet.

The standard TinyOS call to send a packet is called `send()`, and takes as parameters the address of a neighbor, a pointer to a message to send, and the message length:

```
command error_t send(am_addr_t addr, message_t* msg, uint8_t len);
```

We extend this in the anycast case to a new call, `sendany()`:

```
command error_t sendany(am_addr_t* addrs, uint8_t n, message_t* msg,
                              uint8_t len, ers_t ers),
```

---

[5] These constants were chosen based on the informal observation that many "good" wireless links approach 100% packet delivery at short timescales; a detailed evaluation remains to be done.

where the key differences are the passing of a *list* of neighbor addresses, as well as a value encoding the type of ERS that is requested by the network layer. This entirely decouples the working of the anycast forwarding at the link layer from the network layer computation of anypath routes. For example, in our implementation, the link layer decides based on the nodes in the destination list, and on the σ_i values in its neighbor table, whether to use sender-driven (synchronous) forwarding or receiver-driven (asynchronous) forwarding. The network layer needs not be aware of this decision; its role is to compute anypath costs and select next hop neighbors.

Note that `sendany` does not specify *how* relay notification is performed when the link layer is doing receiver-driven forwarding. The simplest form of receiver-driven forwarding is achieved by simply using a link layer anycast address (that is defined similarly to the broadcast address); this packet must contain a distance field that receivers of this anycast packet then check to see if they are closer to the destination, making them a candidate relay.

TinyOS is an event-driven programming model, and a call to `send()` is always followed by a `sendDone()` call-back coming up from the link layer:

```
event void sendDone(message_t* msg, error_t error);
```

With anycast forwarding, it may be of use to network layer protocols to know *which* node among the list passed in the `sendany` call was the effective relay. This is indicated with the additional `relay` argument that comes back via the `sendAnyDone()` call:

```
event void sendAnyDone(message_t* msg, error_t error, am_addr_t relay);
```

Of course, the effective relay may not be known to the link layer, in which case it returns the broadcast address.

7.4.3 Network Layer

Moving further up in the protocol stack, we reach the network layer, which computes the anypath cost metric, advertises it to neighbors, and selects candidate relay sets.

Beaconing

Nodes advertise their route costs by means of local, periodic route beacon broadcasts. Each beacon contains the sender address, the destination to which it is advertising a route, the sender's cost to reach that destination, and a destination sequence number that serves to prevent routing cycles.

Computing Costs

Our aim is to decouple as far as possible (that is, without significantly compromising performance) the network and link layers, so that a link layer implementing a different kind of anycast forwarding can replace an existing link layer without forcing the rewrite of all upper-layer functionality. The `sendany` and `sendanyDone` calls achieve this objective for the packet forwarding path; achieving it for the routing plane requires us to abstract the notions of anycast link cost from the underlying anycast forwarding mechanism. Specifically, we require the link layer API to provide a `anycastCost()` call, that takes a list J of neighbor addresses and returns the anycast link cost d_{iJ} to transmit to *any* of these nodes.

```
command uint16_t anycastCost(am_addr_t addr*, uint8_t n).
```

Note that this cost is entirely abstract; the only requirement is that it must be additive; i.e. that it makes sense to add such costs together to obtain the cost of a route.

The network layer itself maintains a neighbor table, that essentially contains the advertised costs of each neighbor from which it has received a beacon. It recomputes its own cost each time a beacon is received, by traversing this list in increasing order of distance, and adding the remaining path cost with the d_{iJ} that is obtained via the call above (as in Section 5.4.2). Given that we use either ERS-All or ERS-Any, the computation of the remaining path cost is equally simple. Having a computationally efficient procedure for updating route costs was essential given the constraints of running the protocol on a simple micro-controller; this goal is achieved with the aforementioned strategy.

7.5 Testbed Evaluation

This section presents some initial performance results obtained from running LARP on a wireless testbed. This testbed, that we designed and installed as part of this thesis work, consists of 50 tinynode modules deployed in various offices and rooms of three adjoining buildings of the EPFL campus.

7.5.1 Testbed Characteristics and Experimental Setup

A networking testbed is only as useful as the data that can be extracted from it. It is essential to have a reliable out-of-band channel to each node, in order not only to reprogram and configure nodes, but also to retrieve monitoring data from each node. In the context of a low-bandwidth wireless network, this necessity calls for the presence of a wired back-channel. Our wireless nodes do not themselves have any built-in facilities to support wired network connectivity. The main wired interface is a serial port. Connecting serial cables to 50 nodes across a building is clearly impractical, and so another solution was required. We chose to use small ethernet-to-serial bridges, namely, the Digi Connect ME [44]. Both modules are IP-capable devices that offer transparent serial port relaying over TCP/IP, allowing a host PC to "mount" the serial port over the network and interact with the remote TinyNode as if it were locally connected to the PC's serial interface. Each TinyNode is physically attached to a Digi module, and we can thus use the building's existing ethernet infrastructure as our backchannel.

Software

Beside the actual hardware setup and the embedded software running on the wireless nodes, we also need a set of intermediate software tools to automate as far as possible the management and configuration of the testbed, and to facilitate the downloading of experiment data from each node.

We developed a tool suite based on the Pytos [107] system of Whitehouse et al. Pytos enables interactive development by allowing a PC to access the functions and variables of the statically-compiled program on a wireless embedded device at run-time.

From a practical point of view, the key feature of Pytos is that it allows easy access from PC-side scripts to functions and data on the wireless nodes, using the Python programming language. Given the `c` code for a TinyOS application, Pytos automatically creates RPC stubs on both the TinyOS and PC sides, allowing to call TinyOS functions and retrieve TinyOS variables transparently from python scripts. This infrastructure eliminates the tedious and error-prone steps of manually writing functions to process, dispatch and marshal RPC-like calls; it offers in a sense the power of being able to attach a debugger to remote embedded devices, using the serial port as a transport channel.

Since our LARP implementation runs under TinyOS 2.0, and Pytos was originally developed for TinyOS 1.x, we ported it forward. This required rewriting or adapting all the functions that automatically generate TinyOS c code, and modifying packet and marshalling formats to be compatible with the newer TinyOS 2.0 packet layouts.

Physical Layout

The nodes in our testbed are deployed over three adjacent buildings, covering a horizontal surface of approximately 2500m^2. Nodes inhabit 4 floors of the buildings, with approximately two thirds of the nodes on the third floor, and an the remainder on the first, second, and fourth floors. Node locations were in large part dictated by practical constraints: each node needed access to an ethernet port, and this ethernet port must be cabled to a specific subnet. Under this strong constraint, we tried to spread out nodes as evenly as possible in all three dimensions; it nonetheless remains that spatial node distribution is far from uniform. Within each office or room, node positions were also dictated by practical constraints, and our primary objective was to place each node in a manner that would least disturb the office's occupant. As a result, some nodes are taped to a wall, some are on the floor, and yet a few others are on window sills. This directly influences connectivity, with nodes placed on the ground having (on average) reduced range compared to wall or desk nodes. The takeaway point is that the physical layout of this testbed is *unplanned*, and node positions were in no way driven by topology concerns.

Topology

We show the testbed topology in Figure 7.5, for two different transmit powers. This topology was obtained by running simple connectivity measurements. Each node transmitted 1000 packets; a link is considered to exist between two nodes when at least 80% of these packets are received in both directions. At 15 dBm transmit power (which is the maximum for the XE1205 transceiver), the network is shallow, and most node pairs are separated by three or less hops (in single-path distance). At 5 dBm transmit power, network diameter increases, with some paths having up to 6 hops. Three nodes ($37, 38$, and 49) are also disconnected from the main cluster. We also show in Figure 7.6 a histogram of node degrees.

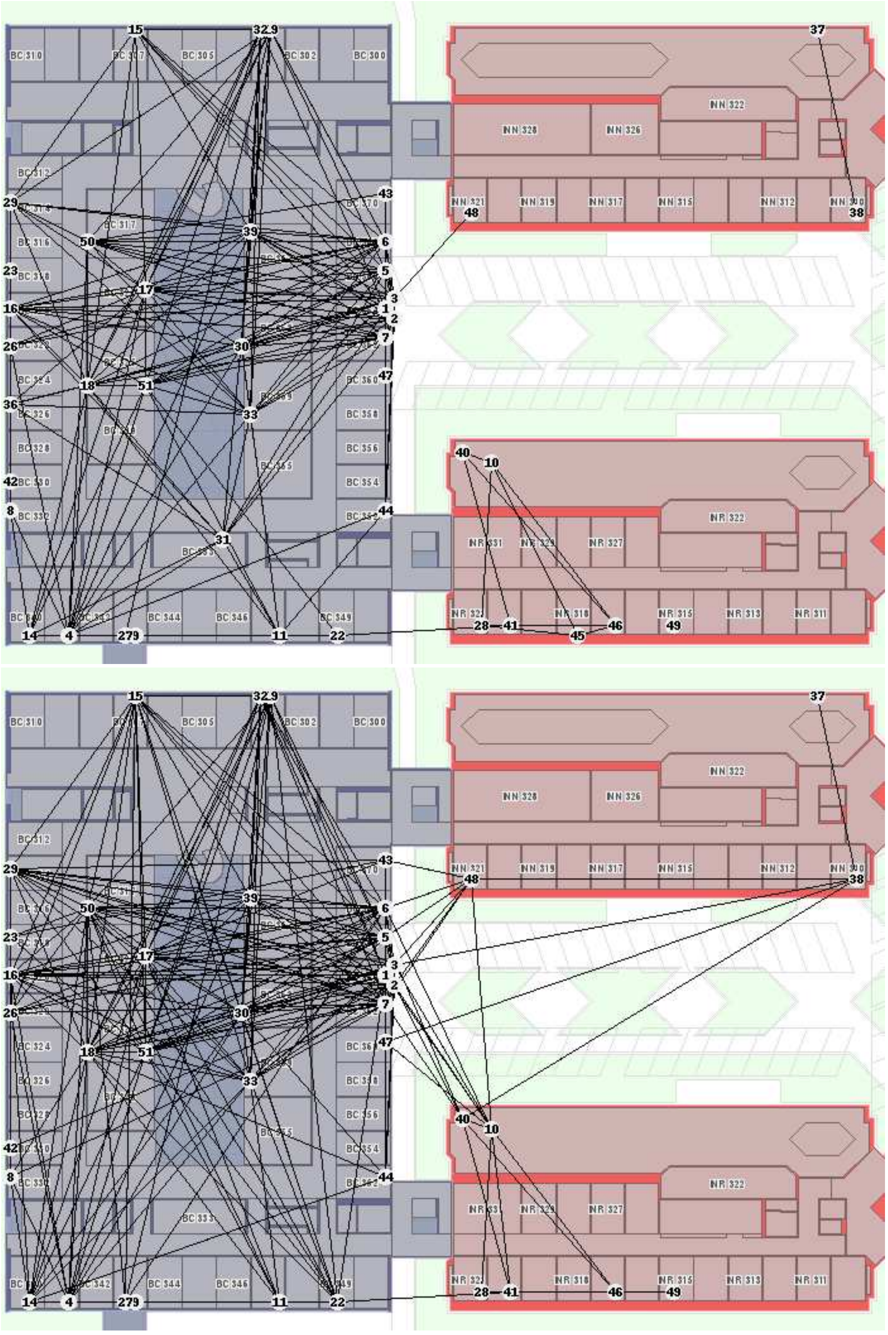


Figure 7.5: Testbed topology at 5dBm transmit power (top) and at 15dBm transmit power (bottom).

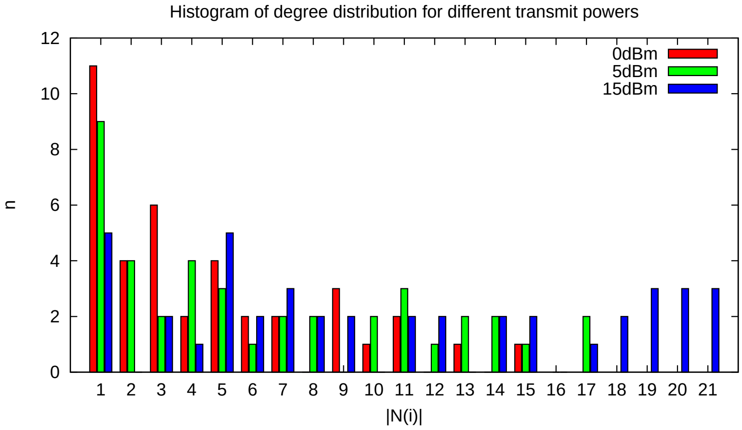


Figure 7.6: Testbed degree distribution for different transmit powers.

7.5.2 Experimental Setup

In this section we present some early experimental results from running LARP on our testbed. Our aim in these experiments was to validate the proper functioning of the protocol implementation, to make a first-order performance assessment, and to compare this performance with that of a single-path routing protocol.

Each of the experiments described in this section was run over varying configurations of transmit power and choice of sink node(s). Specifically, we varied these parameters as follows:

- **Transmit power:** We ran one set of experiments with all nodes using 5dBm transmit power, and one with 15dBm transmit power.
- **Sink nodes:** We used three different sink configurations. Two configurations had a single sink (node 14, in the SW corner of the plan of Figure 7.5, and node 48, in the INN building that is in the NE portion of the plan). The third configuration employed both nodes 14 and 48 as sinks, in order to exercise the network-layer anycast functionality of LARP.

The combination of two transmit power settings and three sink configurations gives us a total of 6 experiment setups. Each setup was run once with anypath routing, and once with single-path routing, each time for three hours, giving a total running time of 36 hours. For single-path routing we used the same protocol, but with the network layer constrained to select candidate relay set sizes of size 1. This allows a fair comparison between anypath and single-path that was not affected by protocol implementation differences.

Nodes ran at a 0.8% duty cycle, giving a wakeup period $t_{rx} = 300ms$, and a listen time of $t_l = 2.12ms$. Data packets were originated at an average rate of one packet per minute at each node. Routing beacons were transmitted every 10 minutes, except for the first 5 minutes of each three hour run, where routing beacons were transmitted every 30 seconds, in order for routes to converge more rapidly. Note that with these data and beacon rates, hybrid anycast forwarding rarely resorted to receiver-driven asynchronous transmission, because schedule synchronization for a large majority of node pairs remained sufficiently tight that this was not necessary.

7.5.3 Results

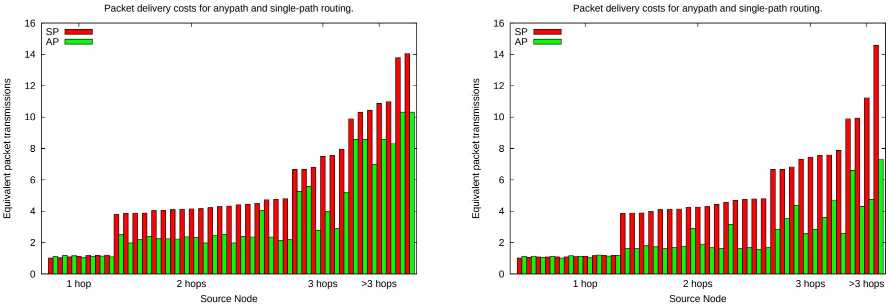
We instrumented the XE1205 radio stack to keep records for the last 20 packet transmissions. For each outgoing packet (including both locally originated and relayed packets), our instrumented code logged the originator address, a 16-bit sequence number placed by the originator in the packet payload, and the total number of transmitted bytes. This last value includes *every* transmitted byte, including both the preamble and the packet itself. It therefore allows us to account for and compare the costs of variable-sized preamble lengths that result from nodes having different sizes of candidate relay sets and different σ_i associated with neighbor schedules.

We used Pytos scripts to periodically download the contents of the instrumented log queue to a PC for offline analysis, effectively giving us a trace of all transmitted packets and their cost. Since each relayed packet can be uniquely attributed to a given source, we were able to infer the total number of bytes transmitted across the network for each originated packet. For ease of comparison, we converted the total number of transmitted bytes into equivalent packet transmissions. In other words, if one originated packet resulted in k transmitted bytes across multiple hops, the number of equivalent packet transmissions is $\frac{k}{l}$, where l is the length (in bytes) of a packet and its associated physical-layer framing[6]

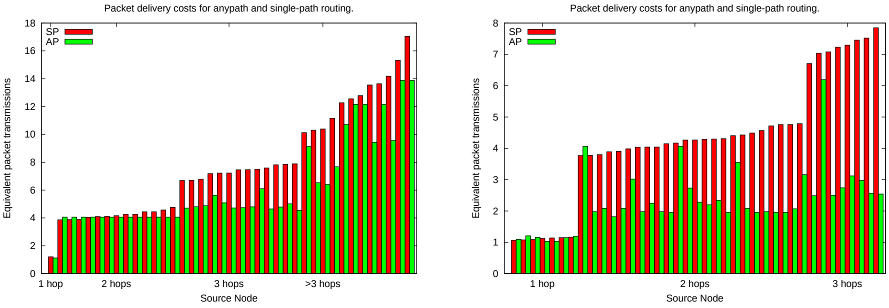
This data is represented in Figure 7.7, with one plot for each of the 6 experiment configurations described above. Each bar in a plot represents the total equivalent packet transmissions, averaged over all packets originated by a given source node.

---

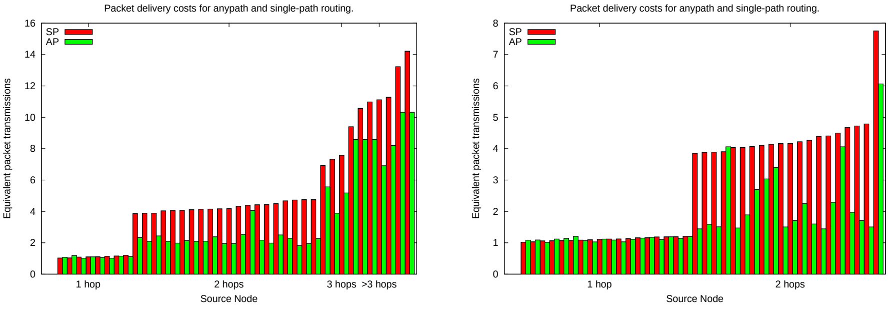
[6] More precisely, we count the physical-layer framing for a transmission to an always-on receiver, which with our XE1205 radio driver consists of 4 preamble bytes and a 2-byte sync word.



(a) Transmit power 5dBm, sink node 14. (b) Transmit power 15dBm, sink node 14.



(c) Transmit power 5dBm, sink node 48. (d) Transmit power 15dBm, sink node 48.



(e) Transmit power 5dBm, sink nodes 14 and 48. (f) Transmit power 15dBm, sink nodes 14 and 48.

Figure 7.7: Average packet delivery costs for single-path (SP) and anypath (AP) routing. Each plot represents one configuration of transmit power and sink node(s). Each bar in a plot represents the *total equivalent packet transmissions*, averaged over all packets originated by a given source node. Nodes are ordered by increasing single-path cost.

Nodes are ordered by increasing single-path cost. Average single-path hop distances are also annotated along the x-axis; the "step" between the costs associated to successive hop-counts is less and less marked as distance increases, due to the fact that this resulting cost is the sum of an increasing number of components with a random part. Unsurprisingly, overall costs are lower at 15dBm than at 5dBm, and they are lower with two sinks than with one sink.

Most importantly, these plots show that anypath routing decreases costs for all nodes that are more than one hop away from the sink(s). Only two source nodes have higher cost to reach a sink with anypath than single-path (in Figures 7.7(d) and 7.7(f)); for packets originated at other nodes (at distances greater than one hop from a sink) anypath routing requires less overall transmissions by a factor of up to 3.

Figure 7.7 takes into account only the transmission cost for packets that were successfully received at a sink. However, neither routing protocol achieves perfect reliability, and so it is necessary to also consider the end-to-end delivery rates. We show in Table 7.4 the minimum, average, and maximum packet delivery rates over all testbed nodes for each experiment. All single-sink experiments had one or two "straggler" nodes with large numbers of lost packets, e.g. resulting in a delivery ratio below 85%. Aside from these poorly connected nodes, overall delivery rates are satisfactory, leading us to conclude that at least for this particular testbed, our simple link estimator makes generally correct decisions on which links are reliable enough to be used. We show in Figure 7.8 two route snapshots from these experiments.

Sink(s)	TX Power	**Min**		**Average**		**Max**	
		AP	SP	AP	SP	AP	SP
14	5dBm	72.2%	51.4%	92.1%	94.2%	100.0%	99.4%
14	15dBm	85.3%	81.1%	96.1%	88.1%	100.0%	100.0%
48	5dBm	48.8%	61.6%	92.4%	90.1%	100.0%	98.8%
48	15dBm	58.0%	71.7%	94.5%	86.3%	98.2%	99.4 %
14,48	5dBm	94.8%	93.5 %	88.9%	91.2%	99.4%	100.0%
14,48	15dBm	93.3%	89.1%	96.7%	91.1%	100.0%	98.8%

Table 7.4: Packet delivery rates for single-path (SP) and anypath (AP) routing.

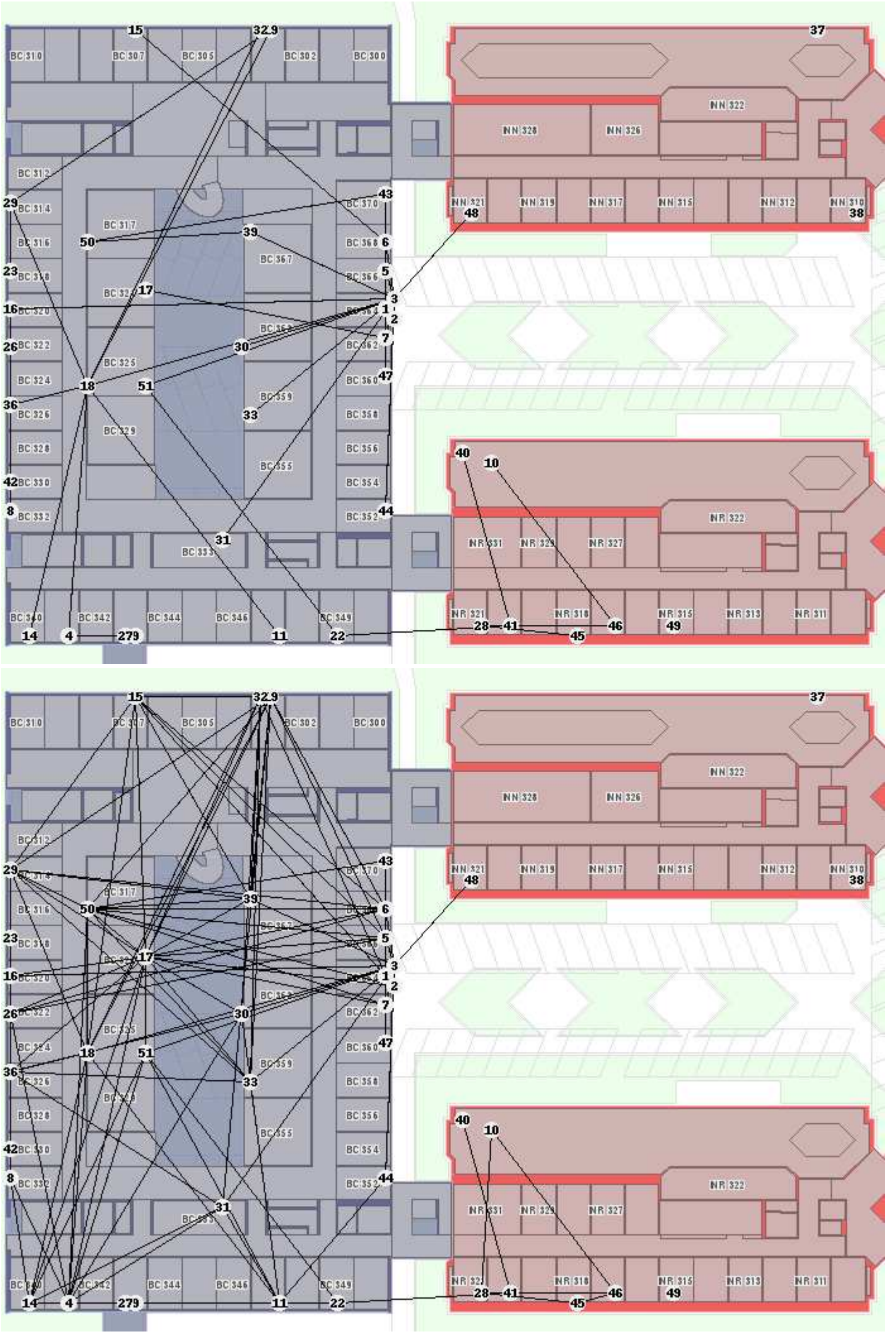


Figure 7.8: Route snapshot with destination node 48 and 5dBm transmit power. Top: single-path route, bottom: anypath route.

7.6 Summary

In this chapter, we have described the design and implementation of a complete anypath routing protocol for low-rate, low-power multi-hop wireless networks. Evaluation on a 50-node testbed indicates that anypath routing is robust, stable, and increases energy efficiency of low-power nodes by a significant factor over the equivalent system using single-path routing.

Chapter 8

Multi-hop Error Correction with Packet Combining

The previous chapters have viewed communication at the *granularity of a packet*: either a packet is received entirely, or it is entirely lost. However, some of the packets that we previously viewed as "lost" may in fact be received, but with a few bit errors that cause them to be dropped by (software or hardware) decoding mechanisms at the link or network layers.

Considering packets as our finest-grained unit of information ignores these situations, and so in this chapter we step down to a lower level of granularity, and "peel open" packets to take into account what happens at a bit level.

One option to increase the reliability of links prone to bit errors is forward error correction (FEC): each packet is sent with some redundant bits allowing to correct (a limited number of) errors. The class of error-correcting codes is vast [67] [10], and many have been applied to wireless communications. These instances have typically addressed point-to-point communications. In other words, they work over *a single link at a time*. But in a multi-hop wireless network, interactions may happen across several links at a time. A novel aspect of the scheme proposed in this chapter is that it takes advantage of these multi-hop interactions by using *overheard* packets at nodes other than the packet's destination. For example, when a node sends a packet to the next hop in a route, it is possible that an upstream node already receives some of the bits in the packet. Traditional error-correction techniques are unable to exploit such information that arises from multi-hop interactions.

To answer this question, we have designed, implemented, and measured *Simple Packet Combining (SPaC)*, a cooperative diversity system that performs error correction by combining corrupt packets. The proposed design is primarily targeted to the class of low-rate sensor networks where nodes sleep most of the time and where channel utilization is very low. Packet loss in such networks is primarily caused by fading and attenuation rather than congestion and collisions. Consequently our design is geared toward corruption consisting of small numbers of errors rather than packets with long error bursts due to interference.

Nodes buffer corrupt packets upon reception, and when two or more corrupt versions of a packet have been received, a *packet combining* procedure attempts to recover the original packet from the corrupt copies. If these corrupt copies correspond to identical transmissions of a same packet, then combining them corresponds to decoding a repetition code. With three corrupt packets, repetition decoding can be done by a simple majority voting scheme. With only two corrupt packets, we use a different scheme called *merging* and for which we develop an efficient implementation based on incremental CRC computation. A repetition code has weak error-correcting power; it is preferable when possible that multiple corrupt copies have different encodings, so that combining them becomes a *decoding* operation with greater error-correcting power. Specifically, our system uses a systematic, invertible block code, and transforms some packets into *parity* packets with this code before transmission. Note that the original bits can be recovered from a parity packet if it is received without errors, and that a parity packet has the same length as the original. Therefore, a noteworthy aspect of the scheme is that it never imposes redundant overhead on transmissions which are received without errors.

On point-to-point links, packet combining behaves similarly to a class of techniques known as Hybrid ARQ (HARQ), that originated in the work of Sindhu [94]. SPaC further generalizes HARQ to multi-hop settings using a novel form of error correction that exploits *overhearing* packets from the broadcast wireless medium. As such, SPaC is a *transparent* extension to the link layer that increases the efficiency of upper layer protocols. It works either in conjunction with link-layer retransmissions, if those are enabled, or without link-layer retransmissions.

We report on experimental results for single-hop with retransmissions, single-path routing, anypath routing, and routing with hop-by-hop retransmissions. Gains increase automatically as one or more of the underlying links traversed becomes more and more lossy. Broadcasting experiments and integration into a real deployment further show that the gains are highly dependent on the underlying links and topology. Because packet combining transmits no overhead, it never decreases performance, allowing it to be deployed uniformly through a network without penalty.

We also examine the energy cost of CPU processing. Profiling shows that computation energy cost cannot be entirely be neglected in comparison with communication cost, contrary to common assumptions.

Summary of contributions

We are not the first to propose using packet combining in conjunction with multi-hop network primitives. However, there is a sizeable gap between the design and analysis of coding schemes in abstract terms, and their realization in a running system. To the best of our knowledge, this work is the first to design and implement a complete system for multi-hop wireless networks. Specifically, the contributions of this chapter include:

- Designing a complete error-correction system using multihop packet combining.
- Going from this design to a full implementation running on low-power, low-complexity wireless sensor nodes. This requires a careful investigation of preamble detection mechanisms, and motivates a new method for incremental CRC with that recomputes a CRC on a packet with *one operation* per changed bit.
- Evaluating the performance benefits of this packet combining on three separate wireless testbeds.

This chapter is organized as follows. Section 8.1 gives an overview of packet combining in different situations. Section 8.2 shows detailed bit-level measurements indicating that many errors can be corrected with simple channel codes, and introduces an error-tolerant preamble detection scheme motivated by these measurements. A detailed presentation of SPaC is in Section 8.3, Section 8.4 discusses our implementation. Experimental performance, results are given in Section 8.5, and a summary in Section 8.6.

8.1 Overview

Before entering technical specifics, we explain how packet combining comes into play under three networking primitives. We only assume at this point the existence of an algorithm that can recover (with some probability) the original packet from two or more corrupt copies. The network primitives used to illustrate packet combining are single-hop, multi-hop routing, and broadcasting. These are also the ones used for the micro-benchmark evaluation of Section 8.5.

Other networking primitives can also benefit from packet combining, for example multicast, anycast or multi-path routing. The examples presented here are chosen to be simple and general, and to capture a broad class of networking primitives

Primitive	Nodes that can exploit received corrupt packets.
Single-hop	Destination
Multi-hop	All nodes on route between sender and destination
Broadcast	All nodes

Table 8.1: Interaction spans.

present in sensor networks. Note that further protocol-specific are possible for each individual primitive. For example multi-hop routing may be improved by an end-to-end reliability mechanism; and flooding can be made more efficient by duplicate suppression. We should emphasize that packet combining in no way precludes the use of such protocol-specific optimizations, but rather that it is an underlying link layer mechanism that applies transparently to upper layer protocols.

8.1.1 Examples

Single-hop packet combining. In a standard ARQ system, the receiver immediately discards a packet received with errors. Hybrid ARQ (HARQ) schemes improve upon ARQ by buffering a corrupt packet at the receiver whilst awaiting retransmission, and combining multiple corrupt copies to do error recovery. The idea of HARQ originated in the work of Sindhu [94], who first considered the idea of merging two non-coded packets to correct errors. Rather than retransmit the original packet as is, the sender may retransmit the parity bits produced by applying an encoding operation to the original packet. This allows the packet combining decoder to recover from more errors than if the same bits had been transmitted twice. The idea of using plain and parity packets, which we use here, has been studied in various forms using increasingly complex codes [67] [19]. The contribution of this work is to generalize the ideas of hybrid ARQ to multi-hop primitives and apply them in a practical setting.

Multi-hop packet combining. When a packet traverses a multi-hop route, HARQ may happen on a hop-by-hop basis anywhere along the path. But beyond this, packet combining also enables a novel form of multi-point interaction that occurs whenever an upstream node beyond next hop overhears a corrupt packet. This corrupt packet is buffered and used for error correction when the next hop forwards the packet. By exploiting this multi-point interaction in addition to single-hop combining, the effect of packet combining on a route is greater than the sum of packet combining on constituent hops.

The top row of Fig. 8.1 depicts a three-hop route with a sender A, relays B and C, and destination D. In Fig. 8.1(b), the sender transmits a packet to B, who receives it without error. Though the link from A to C is too lossy to be used by the protocol,

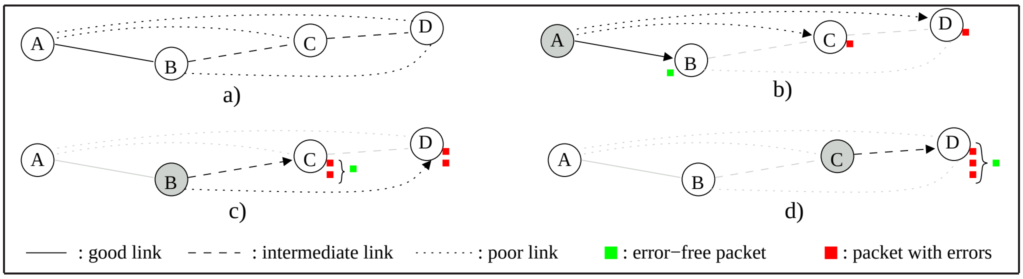


Figure 8.1: Packet combining with multi-hop routing.

it still delivers a large number of corrupt packets, and C receives a corrupt copy of this packet. In Fig. 8.1(c), node B forwards the packet, which is received with errors at C. Since C now has two corrupt copies of the packet (one sent by A, one by B) it can combine them and (with some probability) recover the original packet. Finally C forwards the packet which arrives with errors at D, who now has three corrupt copies and recovers the packet by combining them. Note that C may occasionally overhear A's packet without errors. The use of an anypath routing protocol that can exploit such packets is orthogonal to packet combining, and both may be used in complement. We should emphasize that *no network layer state* need be exposed to the packet combining layer; this layer simply accepts corrupt packets blindly and attempts to combine them, without using topological information from the network layer.

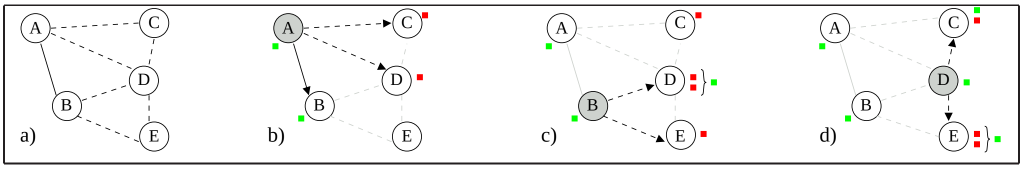


Figure 8.2: Packet combining with network broadcasting.

Broadcast Packet Combining. We now turn to the network broadcast primitive, where one node seeks to disseminate a packet to all others. We consider a simple flooding protocol that forwards each flooded packet once (with duplicate detection by a sequence number or random identifier). Other, more efficient approaches to broadcasting are possible [65] [40]. We note that packet combining can enhance other broadcasting protocols in a manner similar to flooding, and study flooding because it is a simple mechanism found in many applications (such as Surge, tinyDB [69], or directed diffusion [43]).

Fig. 8.2(a) shows a 5-node topology. In Fig. 8.2(b), node A originates a flood packet that is received without errors by B, and with some errors by C and D. B now transmits the packet (Fig. 8.2(c)); it is received with errors by nodes E and D. Node D now has two error copies of the same packet, from which the packet combiner decodes the original packet. Finally in Fig. 8.2(d), node D transmits the packet. Node C receives the packet without errors, and can therefore discard the corrupt packet held in its buffer. Node E however receives a corrupt packet, and combines the two previous corrupt copies to correct the errors.

Unlike multi-hop routing, flooding never uses link-layer retransmissions, and therefore cannot benefit from the single-hop interaction of HARQ. However packet combining can take advantage of a larger number of bad packets in flooding than routing, since a bad packet received *at any node* may be exploited (Table 8.1).

8.1.2 Comparison with FEC

An alternative solution to packet combining would be to use standard FEC techniques. It is therefore natural to ask: *Why not use FEC instead of packet combining?*

The answer is two-fold. The first part concerns link variability. For any link with known and stationary error characteristics (bit-error rate, burstiness), one can design a channel code which optimally matches those characteristics. However, as soon as the link deviates from the characteristics the code was designed for, performance drops sharply: if the link is more error-prone than expected the code cannot recover the errors, and if it is better than expected, transmitting parity bits becomes redundant overhead. In particular, appending *any* error-correcting bits to a packet which arrives without errors is highly wasteful. Sensor network measurement studies [114] [13] have shown that link variability is high both in time and across space, in particular for those very links which are error-prone and most require FEC. Therefore, it is difficult to design an efficient static FEC system for the wide range of channel conditions seen in a sensor network.

Adaptive FEC can provide a solution to time-varying channel conditions, but requires frequent, costly channel measurements to obtain a timely and accurate estimate of the channel bit-error rate. These measurements are especially costly when the traffic rate is low and the channel potentially changes between every packet transmission. In comparison, packet combining provides a simple form of adaptive channel coding that requires no explicit channel measurements: if the channel is good, the initial transmission is received correctly with no redundant error correction bits having been transmitted. If the channel is poor, then the retransmission needs only to contain additional error-correction bits to “elevate” the two combined packets at the receiver to a lower-rate code, with which more errors can be corrected.

The second part of the answer is that packet combining naturally transposes to multi-hop scenarios, whereas FEC is inherently geared toward point-to-point links. In the case of multi-hop routing, FEC can improve the performance of each individual links, but can not take advantage of *overheard* corrupt packets at downstream nodes. In contrast, packet combining can do both. In the case of broadcasting, a node receiving multiple corrupt copies from different nodes cannot combine them with standard FEC.

8.2 Channel Measurements

Before designing any error correction system, it is necessary to know the error characteristics of the channel being addressed. In the case of packet combining, we are interested in the following two aspects:

- **Error characteristics:** What bit error rates do we have in corrupt packets? How bursty are error patterns? We will see that error characteristics are such that a simple channel code can already correct a large number of error patterns.
- **Sources of packet loss:** What portion of packet loss is due to packets that are received corrupt and discarded by the link layer, and what portion is due to *missed packets*, i.e., those packets that were not received at all? This breakdown is critical: if loss is primarily attributable to missed packets, then an error-correction scheme will see too few corrupt packets to be effective. These measurements motivated the design of a new error-tolerant preamble detection scheme (ETPD) that we present in the final part of this section.

We consider an asynchronous, random-access channel. A known preamble pattern, serving to achieve receiver synchronization, precedes each transmitted packet. A node in receive mode continuously draws in bits from the radio until the previously received bits *exactly* match the preamble sequence; once a preamble has been detected the full packet is drawn in. If a preamble is received with any bit errors, then the entire packet is missed[1]. The notations are as follows:

- p_d is the probability of **error-free packet delivery**.
- p_c is the probability of **corrupt packet delivery** (packet received with one or more bit errors).

---

[1] This mechanism is used in 802.11 and 802.15.4 as well as non-standard radios like the CC1000.

- $p_m = 1 - p_d - p_c$ is the probability of **missed packet** (packet not received at all, because the preamble was mis-detected). We therefore have $p_m + p_d + p_c = 1$, and the overall packet loss probability is $p_c + p_m$.
- $R_{cm} = p_c/p_m$ is the **ratio of corrupt to missed packets**. Packet combining will work better with high values of R_{cm}; the ideal case is when p_m is close to 0.
- L_{pre}, L, and $L_{tot} = L_{pre} + L$ are respectively the preamble length, packet length (including headers and payload), and total transmission length, in bits.
- $\hat{p}_d$ (resp. $\hat{p}_c, \hat{p}_m$, and $\hat{R}_{cm}$) denotes the estimated value of p_d (resp. p_c, p_m and R_{cm}) made on the empirical data for a given link.

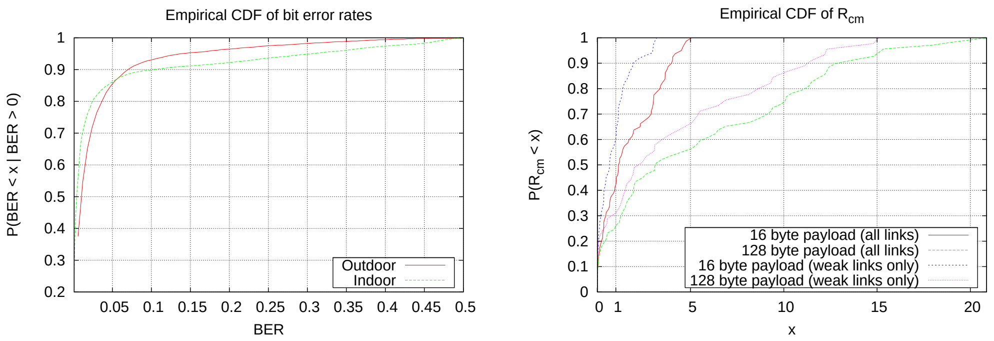
We used two testbeds: a 39-node indoor testbed with nodes attached to the ceilings across several offices on one floor of UCLA's Boelter Hall and a 10-node outdoor testbed installed in a UCLA courtyard. Each node is a Crossbow Mica2, which has an Atmel ATmega128L microcontroller with 4KB of RAM, 128KB of Flash, and a CC1000 radio [14]. The radio operates in the 433Mhz band, uses narrowband 2-FSK modulation, and runs at a bit rate of 19.2Kbps (bits are Manchester-encoded, resulting in a 38.4kbps symbol rate). In our experiments each node was in turn the sender; other nodes listened and logged received packets via a wired backchannel. We ran experiments varying two parameters: transmission power (-15dBm and -5dBm) and packet length (16 and 128-byte payloads, with a 5 byte header and a 2 byte trailer), giving us a total of four configurations and 150,000 packets transmitted.

Error characteristics are very different on a congested network, where a large number of errors are due to interference bursts from concurrent transmissions, and on a network with low channel occupancy. Our aim is to improve performance of low-power, low-rate applications, and we consequently made measurements on a "silent" network with no concurrent background traffic. Note that in the presence of external RF sources (other wireless devices, electronics), burst interferences are still possible even in a silent network.

8.2.1 Bit-level Error Measurements

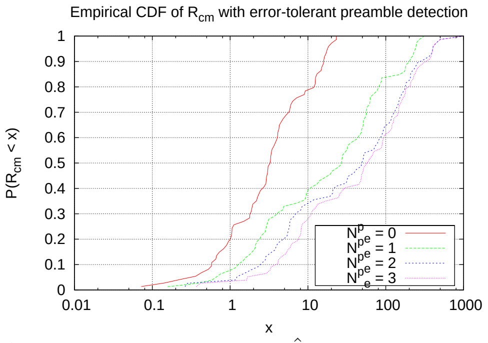
We now examine the error characteristics of our channel, starting with the number of bit error rates observed in corrupt packets. Figure 8.3(a) shows the distribution of BER over all corrupt packets. Most corrupt packets have few bit errors: the proportion of corrupt packets with a bit error rate below 0.05 is 85% both for short and long packets. The proportion of packets with a bit error rate greater than 0.1 is 6% for short packets and 10% for long packets.

With two or more corrupt copies of a packet, we effectively have an encoding that



a) The empirical CDF of bit error rates, for all packets with at least one error. Most corrupt packets have few bit errors.

b) Empirical CDF of $\hat{R}_{cm}$, the ratio of corrupt to missed packets. "Weak" links are those with $\hat{p}_d < 0.4$.



c) Empirical CDF of $\hat{R}_{cm}$ with error-tolerant preamble detection (ETPD).

Figure 8.3: Bit error rate and preamble detection measurements.

is half-rate or lower. Bit error rates of 0.05 are within corrective reach of half-rate codes [67]. This indicates that many corrupt packets should be recoverable through packet combining. But bit error rate alone does not completely characterize the error process; in particular burstiness is an important factor to guide the choice of error-correction mechanism. Assume for example the use of a code which can correct one error in every 8-bit codeword, and a BER well below $1/8$. If errors are uniformly distributed, then most codewords will have 0 or 1 errors and will be recovered. For the same BER, but with bursty errors, the probability of a packet containing a uncorrectable codeword with more than one error is increased.

Figure 8.4 shows the empirical bit-error probability conditional on an error having occurred K bits earlier. Horizontal lines show the average bit-error rate for each data set, or equivalently, the auto-conditional error probability of an i.i.d. error process

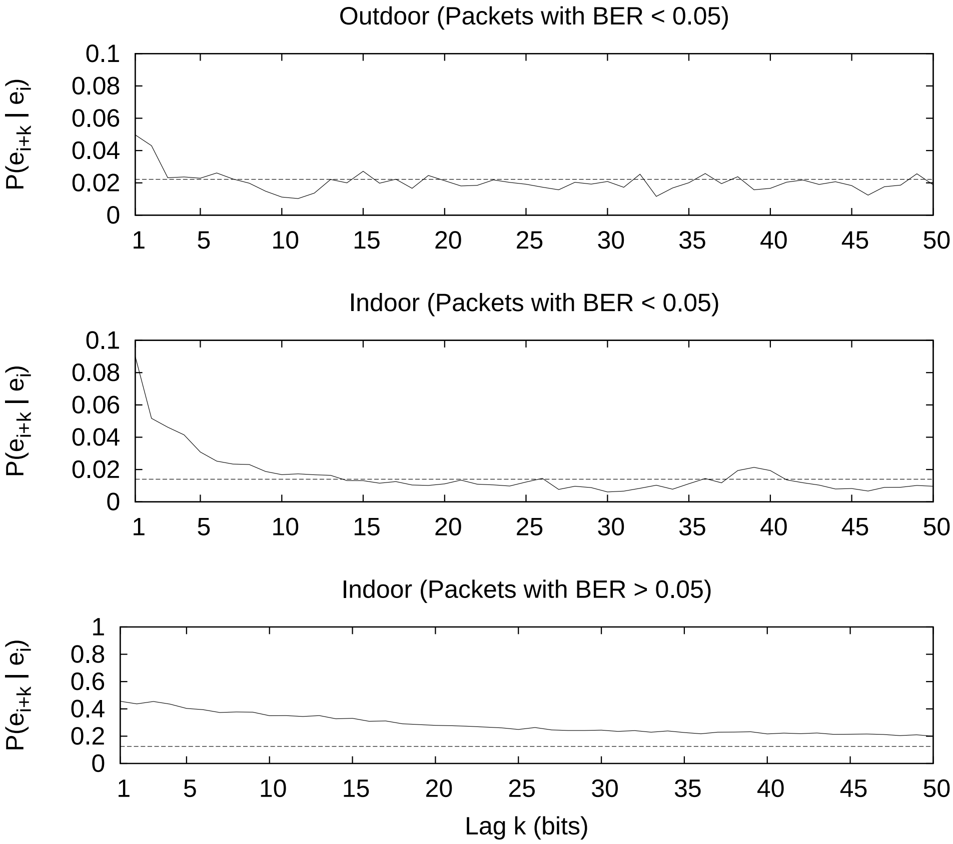


Figure 8.4: Empirical bit-error probability conditional on an error having occurred k bits earlier. Horizontal lines show the average bit-error rate for each data set. Top and center: packets with $BER < 0.05$. Bottom: packets with $BER > 0.05$.

with same BER. These plots suggest that one can coarsely classify corrupt packets into two categories: packets with a low bit-error rate and low error burstiness, and packets with higher error rates and a bursty error process. Since the former category represents nearly 90% of all corrupt packets, (see Fig. 8.3(a)) and since the latter has higher error rates which are harder to correct with low computational complexity, we design Section 8.3 an error-correction coding which is designed to correct the low BER majority of packets, but not the highly bursty ones.

The top and center plots are computed over the subset of packets with $BER < 0.05$. Of these two plots, the outdoor is least bursty: the presence of an error at bit i slightly increases the probability of errors in the following two bits, but after that the error probability is the same as for an i.i.d. process. The indoor trace is slightly more bursty, with the probability of an error immediately following a previous one at approximately 0.09. This difference between indoor and outdoor burstiness, may be due to interfering RF sources which are more likely to be found inside a building than outside. The bottom plot is computed over the much smaller subset of packets with $BER > 0.05$. Here, the picture changes sharply: the error process is much more bursty, over a range of at least 50 bits. For these packets, the short code discussed above would not suffice, since the probability of at least one codeword in the packet having multiple errors is high.

While we have chosen, due to complexity constraints, to "ignore" the class of packets with highly bursty errors, we note that two simple techniques are possible two improve error correction with bursty errors. The first is interleaving. The second is to use codes with block lengths larger than the burst lengths. Unfortunately, it is not feasible to implement either technique in software with simple microcontrollers. For increased burst-tolerance and robustness, both interleaving and longer-block decoding are possible in a hardware implementation.

8.2.2 Sources of Packet Loss and ETPD

We now turn to the breakdown of packet loss between corrupt and missed packets. Fig. 8.3(b) shows the empirical CDF of $\hat{R}_{cm}$, the observed ratio of corrupt to missed packets, for short and long packets. We observe that $\hat{R}_{cm}$ is lower for short packets than for long packets. For example no links have $\hat{R}_{cm} > 5$ with short packets, whereas with long packets over 40% of our links have $\hat{R}_{cm} > 5$. This observation can be explained by noting that p_m is independent of packet length (because preamble length is constant), whereas p_c increases with packet length, since for a given BER, the probability that at least one bit in the packet is corrupt increases with the number of bits. From this observation, we expect packet combining to give more gains with long than short packets.

Fig. 8.3(b) further distinguishes $\hat{R}_{cm}$ over weak links, which we defined as those with $\hat{p_d} < 0.4$. $\hat{R}_{cm}$ is lower when considering only the weak links. Yet, links with a low delivery rate are those where packet combining is most needed, and if packet combining sees too few corrupt packets its utility will be limited. Since a receiver misses a packet whenever its preamble is received with errors, these measurements motivated the design of ETPD, to increase R_{cm} by allowing the reception of a packet in the presence of preamble bit errors.

Our initial implementation of ETPD accepted preambles with up to $N^p_{max} = 4$ bit errors (out of 4 preamble bytes). We set this initial value for N^p_{max} because packets with more preamble errors are likely to have too high a bit error rate to be corrected with a half-rate code, and furthermore allowing a greater number of errors would increase the probability of false positives. We added the additional constraint that no single byte could have more than 1 bit errors, to further reduce the probability of false positives and allow for more efficient software implementation.

We ran experiments identical to those of Section 8.2.1, except that the received preamble was also dumped out along with each packet. We then classified received packets according to the number N_e of preamble bit errors. Figure 8.3(c) compares the empirical CDF of $\hat{R}_{cm}$ with and without ETPD for $1 \leq N_e^p \leq 4$. The data for all CDFs originate from the same packet traces: by examining the received preambles, we can see which packets would be received with a given preamble error-tolerance. ETPD allows a significant increase in $\hat{R}_{cm}$: we go from having $\hat{R}_{cm} > 10$ for only 20% of links without ETPD, to 60-70% with ETPD (depending on the setting of N_e^p).

The observed increase in $\hat{R}_{cm}$ does not demonstrate alone that ETPD is worthwhile. Under the assumption that the channel does not vary dramatically over the duration of a packet, we expect that packets with error(s) in the preamble will have a higher bit-error rate than those without. If those packets have a sharply increased BER, then receiving them may be pointless. We therefore classified corrupt packets by the number of preamble errors each one had, and computed in Table 8.2 the observed bit-error rate for each set of corrupt packets. This table shows that the packet BER increases with the preamble BER, in line with the intuition that BER does not vary widely over the duration of a packet.

Our final ETPD implementation can be configured for maximum allowed preamble errors N_{max}^p between 0 and 3. We set $N_{max}^p = 2$ for all further experiments reported in this chapter, since with higher error tolerance, the marginal increase in packet yield is small (Fig. 8.3(c)), and the additional packets have a very high bit error rate (Table 8.2).

A final aspect to consider with ETPD concerns the increased probability of false positive preamble detections. When placed in receive mode, a radio transceiver continuously reads and demodulates bits from the RF front-end. If no transmission is ongoing, this effectively amounts to reading random bytes. By increasing the preamble detection's error-tolerance, we increase the probability of having a sequence of random bytes that is falsely detected as a preamble. We must therefore take into account the probability of spurious packet receptions due to false positive preamble detections. The probability of a spurious reception $P(N_e^p)$, when tolerating exactly N_e^p errors in the preamble can be simply computed by counting the number of possible 32 bit sequences with N_e^p errors, and no more than 1 error per byte: $P(N_e^p) = 2^{-32}\sum_{i=0}^{i=N_e^p} \binom{N_e^p}{4} 8^{N_e^p}$. For our chosen setting $N_{max}^p = 2$, this evaluates to a probability on the order of 10^{-7}, meaning that the number of spurious packet

N_e^p	Preamble BER N_e^p/L_{pre}	Packet BER $(L=16)$	$(L=128)$
0	0	0.029	0.032
1	0.031	0.052	0.084
2	0.063	0.073	0.128
3	0.094	0.095	0.165

Table 8.2: Observed bit-error rate as a function of number of preamble bit errors N_e^p, for two different payload lengths L.

receptions occurring in practice is negligible.

8.3 Decoding and Merging

Section 8.1 showed how a node may come to receive corrupt packets in various settings. We now describe the packet combining algorithms used on the receive path and the corresponding packet encoding functions on the transmit path.

8.3.1 Linear Block Codes

The class of error-correcting codes is vast. We focus in this work on *linear block codes*, which can be implemented efficiently in software using table-based methods (unlike, for example, convolutional codes). A (n, k) linear block code is defined by its generator matrix G, of size $k \times n$. The encoder multiplies each input block with the matrix G, transforming k input bits into a codeword of length n. The decoder, given a received codeword of n bits, finds the closest codeword (in the set generated by G) and returns the information bits corresponding to that codeword. If the number of bit errors is higher than the half-distance between two codewords, then the decoder returns the wrong information bits (that can then be detected with high probability by an outer CRC checksum). If two closest codewords are at equal distance to the received word, the decoder cannot infer which was the transmitted codeword, and declares a failure. For a general overview of decoding algorithms, we refer to [67].

Within the class of linear block codes, we further restrict our attention to *invertible* and *systematic* linear block codes. A block code is systematic if the first k bits of a codeword are the same as the input message bits, in other words the generator matrix of a systematic code is of the form $G = [I|P]$ where I is the $k \times k$ identity matrix and P is a $k \times (n-k)$ matrix called the parity matrix. A block code is invertible if $n = 2k$ and the matrix P is invertible. Note that an invertible code is half-rate since $n = 2k$. Given this matrix P, and an input word m (of length k bits), the encoder outputs the $2k$-bit codeword $Gm = [I|P]m = [m|Pm]$. This codeword is the concatenation of the unmodified input m, which we call the **plain** bits, and Pm, which we call the **parity** bits. Note that since P is invertible, there is a one-to-one mapping from Pm back to m, and therefore one can recover the plain bits from the parity bits of a codeword and vice-versa.

8.3.2 Packet Encoding and Decoding

In a standard block-coded FEC system, the encoder transmits the plain and parity bits of a codeword together. This comes at the expense of increasing the number of transmitted bits by a factor of two, in the case of a half-rate code. As discussed in Section 8.1.2, imposing this overhead on every packet transmissions is prohibitive, given that many packets are received without errors and do not need parity bits.

The key intuition allowing one to use a code *without transmitting redundant overhead* is to observe that we can simply send some "plain" packets which are not encoded, and some "parity" transmissions containing *only* the parity part of each codeword. Specifically, consider an input packet $m = m_1 m_2 \ldots m_l$, where each block m_i is of length k bits. The encoder outputs either m unmodified (a **plain** packet), or the packet $m^* = Pm_1 Pm_2 \ldots Pm_l$ (a **parity** packet). How the encoder chooses between outputting a plain or a parity packet is discussed in Section 8.3.4. The motivation for using a systematic, invertible code is now apparent:

1) If a parity packet m^* is received without errors, the original m is obtained simply by multiplying the packet by P^{-1}:

$$\begin{aligned} P^{-1}m^* &= P^{-1}Pm_1P^{-1}Pm_2 \ldots P^{-1}Pm_l \\ &= m_1 m_2 \ldots m_l = m. \end{aligned}$$

Note that since a parity packet has the same length as the corresponding plain packet, the system *does not transmit any redundant overhead on good links*, whether it transmits a plain or a parity packet.

2) Two corrupt copies of a packet (one plain and one parity) can be *jointly decoded* by taking k bits at a time from each packet, and decoding the concatenated word. This operation is illustrated in Fig. 8.5.

Our implementation uses the Hamming $(7, 4)$ code, extended to $(8, 4)$ with an additional error-detecting bit. The $(7, 4)$ code can correct up to one error per 7-bit block; the extended $(8, 4)$ code allows in addition to detect (but not correct) any 2 bit error per 8-bit block. The primary motivation for using the extended code was to have byte-aligned blocks, given that unaligned operations are inefficient to implement in software. The additional error detection capability of the extended code allows the decoding operation to 'abort' as soon as it encounters an uncorrectable error, in which

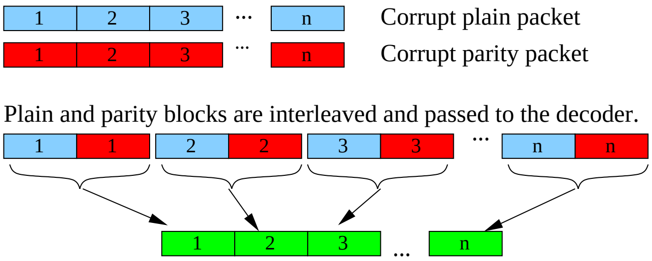


Figure 8.5: Packet Decoding: Recovering the original packet from two corrupt packets, when one is plain and the other is parity.

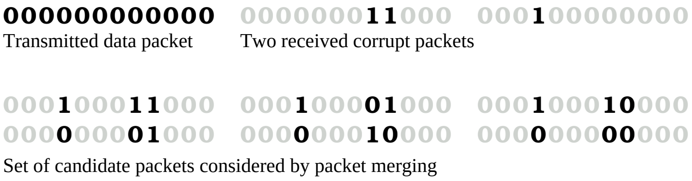


Figure 8.6: Packet Merging: Recovering the original packet from two corrupt packets of same type.

case wasted CPU cycles are saved.

Our choice of a short code was dictated by the practical constraints on decoding overhead: the most efficient software implementation is based on table-lookups, and table size is exponential in block length. With more processing power, syndrome decoding would alleviate these memory requirements. Beyond this constraint, any other systematic and invertible block code can be used with minimal modifications. For example a longer Hamming code, an extended Golay code (with block length 24), or a Reed-Solomon code all have more error-correcting power than the Hamming $(8, 4)$ we used.

8.3.3 Packet Merging

The preceding section showed how two corrupt packets of different types (one plain, one parity) are jointly decoded. What if the receiver has two corrupt packets of same type? We call the corresponding operation *packet merging*. Given that it essentially corresponds to decoding a repetition code, it is not surprising that merging can correct fewer errors than decoding.

Let m_1 and m_2 be corrupt copies of an original packet m; ie $m_1 = m + e_1$ and $m_2 = m + e_2$, where $e_1 \neq 0$, $e_2 \neq 0$, and additional is modulo 2. Note that $m_1 + m_2 = e_1 + e_2$. Therefore by XORing m_1 and m_2, the receiver obtains a *merged error mask* with a 1 in all bits that are errors in either m_1 or m_2. Note that the merged error mask does not show an error bit that occurred in identical positions in both packets. We call such an error a *hidden error*.

Assume that the merged error mask contains n_e non-zero bits. There are then $2^{n_e} - 2$ candidate error patterns that may have occurred. The packet merging procedure corrects each candidate error pattern on one of the corrupt packets, and recomputes the checksum to verify if it matches the transmitted CRC checksum in the packet trailer[2]. Figure 8.6 illustrates a case with $n_e = 3$ and shows all $2^3 - 2$ candidate corrected packets. We distinguish three cases, based on the number k of error patterns for which the checksum is valid:

$\mathbf{k = 0}$. None of the candidate error patterns, corrected on the corrupt packet, yield a valid checksum. Packet merging cannot recover the original packet. This case can only happen if there are hidden errors, since otherwise of the candidate error patterns corresponds to the effective error pattern.

$\mathbf{k = 1}$. A unique error pattern yields a valid checksum. The resulting packet is passed up the stack.

$\mathbf{k > 1}$. Multiple error patterns can recover a packet with a valid checksum. Which (if any) corrected packet is the original packet is undecidable; a failure is declared.

Since the number of candidate error patterns increases exponentially with n_e, the computational overhead becomes prohibitive if we attempt to merge two packets that differ in a large number of bits. The probability of hidden errors increases with n_e, and furthermore, so does the probability of the case $k > 1$ occurring. We can see this intuitively by considering the extreme case where both packets differ in all bits, in which case we would iterate through all 2^L possible packets. For these reasons, we introduce an algorithm parameter n_{max} that upper-bounds the largest value of n_e for which packet merging is attempted. If the merged error mask contains more than n_{max} errors, merging is not attempted and a failure is declared. Besides bounding the number of candidate packets searched, this parameter also controls the two key performance metrics that are probability of merging success, and probability of false positives. We shall see in Section 8.4 that the effective choice in our implementation is dictated by the computational constraints of merging more than the probability of false positives.

For a given set of parameters (bit error rate, packet length), the error correction performance of this merging algorithm is characterized by two measures. The first is the probability of success (the original packet is correctly recovered from two corrupt

---

[2] Merging is not attempted if the two received packets have differing checksums, since this means that either both packets are different (in which case combining is pointless), or that one of the checksums has errors, and we cannot know which (if any) is correct.

copies), and second the probability of false positives (the algorithm produces a 're-paired' packet that is different than the original packet, but for which the checksum is correct). Clearly we wish to maximize probability of success whilst minimizing probability of false positives. Note that even without packet merging, the probability of false positives is never zero, because a CRC checksum cannot detect every possible error pattern. The key is then to ensure that merging does not significantly increase the probability of false positives with respect to a standard receiver.

Merging increases the probability of false positives by at most a factor of $2^{n_{max}}$. Note that in the absence of hidden errors, the error pattern that effectively occurred is present in the set of candidates. Therefore a false positive can only occur in the presence of hidden errors: if there are no hidden errors, and if there is an additional 'false positive' error pattern giving the correct checksum is correct, then the algorithm finds more than one candidate repaired packet and declares a failure (case $k > 1$). This further reduces the probability by a factor of at least n_{max}/L.

Decoding and merging failure probability

We consider i.i.d. bit errors with bit error probability p_e. With the $(8, 4)$ Hamming code which can correct up to one error per codeword, jointly decoding two packets fails if any codeword has more than one error. The decoding failure probability ρ is thus:

$$\rho = 1 - ((1 - p_e)^8 + 8p_e(1 - p_e)^7)^{L/4}. \tag{8.1}$$

where the the exponent is $L/4$ because the decoder operates over two packets of length L bits, or equivalently $L/4$ codewords of 8 bits each.

Merging fails in the presence of hidden errors or when both corrupt copies contain together more than n_{max} errors. For simplicity, we approximate the probability $\tilde{\rho}$ of this event by the probability that two corrupt packets contain less than n_{max} errors, noting that the probability of hidden errors approaches 0 for small values of n_{max}.

$$\tilde{\rho} = 1 - \sum_{i=0}^{n_{max}} \binom{2L}{i} {p_e}^i (1 - p_e)^{2Lpkt-n} \tag{8.2}$$

Efficient CRC computation

The overhead of the CRC computation is critical, because for each candidate error pattern, packet merging must re-compute the CRC on the packet corrected with the error pattern. For example, in MRD [76] the CRC is computed in a non-incremental fashion, and was found to be the bottleneck for the entire combining process. Since the set of bits that change between each candidate packet is small (at most n_{max}), recomputing the CRC over the entire packet is redundant. However, existing incremental CRC algorithms (such as those for ATM/IP networks [11]) do not apply here because they assume that only bits at *fixed positions* (in packet headers) can change at each hop. In our case, the candidate errors can be anywhere in the packet.

We now describe an incremental algorithm that recomputes the CRC on a packet with *one operation* per changed bit. It can be seen as a extension of [11] which removes any constraints on the changed bit's location in the packet. We denote, as polynomials over GF(2), the packet $M(x)$, and $E(x)$ the candidate errors bits (for which we wish to recompute the CRC), we would like to compute the CRC over the message $M^c(x) = M(x) + E(x)$:

$$\begin{gathered} x^r(M(x) + E(x)) \mod G(x) = \\ \underbrace{x^r M(x) \mod G(x)}_{fixed} + \underbrace{x^r E(x) \mod G(x)}_{variable}, \end{gathered} \tag{8.3}$$

where $G(x)$ is the CRC generator. The separation of (8.3) into a sum is possible because the CRC is linear. Defining $E_k(x)$ as the vector containing a single 1 at position k and zeros elsewhere, we can further decompose the variable part of (8.3) as

$$x^r E(x) \mod G(x) = \sum_{i=0}^{L} \mathbf{1}_i (x^r E_i(x) \mod G(x)), \tag{8.4}$$

where the indicator variable $\mathbf{1}_i$ is equal to 1 iff $E(x)$ contains a 1 at bit i. So with a precomputed lookup table T with entries defined as $T[i] = x^r E_i(x) \mod G(x)$, we can recompute the CRC in one operation per changed bit. The table T has length L and can be stored in ROM; in exchange we make a linear gain of L/n_e in computational overhead. As the profiling results of Section 8.4.1 will show, packet merging would be computationally infeasible (in software) without this reduction.

8.3.4 When to Send a Plain or Parity Packet

Decoding can correct more errors than merging, and uses fewer CPU cycles (as we shall see in Section 8.4.1). We therefore wish to maximize the probability that two corrupt packets received successively at a node are of different types (plain and parity), so that they can be decoded. In the case of retransmissions, this is straightforward: the sender alternates between parity and plain. For multi-hop and flooding however,

m	Action
Plain	None
Parity	Invert

a)

m	n	Action
Plain	Plain	Merge
Plain	Parity	Decode
Parity	Plain	Decode
Parity	Parity	Merge, invert

b)

Table 8.3: Receiver processing actions: a) for a valid packet m, b) for two corrupt packets m and n.

there are multiple potential receivers, and the sender cannot know the type of any corrupt packets already buffered at a neighbor. For a node relaying a multi-hop packet, the initial transmission is of opposite type to the last received packet. For a flooded packet, the sender randomly chooses between plain or parity.

Receiver processing depends on the type of packet received. A valid plain packet is passed directly up the stack; a valid parity packet must first be inverted. Two corrupt packets of same type are merged, and then inverted if they are parity; two corrupt packets of different type are decoded. These actions are summarized in Table 8.3.

8.3.5 Extensions

Two simple improvements can be made to the current design. We outline them for completeness and note that they can be implemented as extensions to SPaC. The first is to allow the receiver to buffer more than two corrupt packets. With multiple corrupt copies, the receiver has more candidate pairs to combine. The probability of successful combining thus increases as $\binom{n}{2}$, where n is the number of packets at hand. Note however that this also implies an increase in the worst-case computational cost.

The second improvement is to use a higher order (mn, n) code with a generator matrix of form $G = [G_1|G_2|\ldots|G_m]$, where each G_i is an invertible matrix of dimension $n \times n$. This generalizes the notion of plain and parity packets to m distinct packet types, each being invertible in the absence of errors. A further desirable property is that any two sub-matrices taken together form a half-rate invertible code such as the ones considered previously in this section, so that the combiner can jointly decode any two packets of different types. With all m packets of distinct types, the encoding rate $1/m$ allows to correct yet more errors. The general form of such codes is not known, but some specific examples exist such as the work of Alfaro and Meo [19], who propose a third-order $(24, 8)$ code. Though the 24-bit block length means that decoding three packets jointly would be hard in software on sensor nodes, such a code is advantageous even if we only decode two packets at a time (giving a $(16, 8)$ code), because it decreases likelihood that any two corrupt packets are of same type. With such a code, merging operations are attempted less frequently and the combining success rate is increased.

8.4 Implementation

We implemented ETPD and SPaC in TinyOS and evaluated it with mica2 wireless nodes[3]. The SPaC component currently works over B-MAC, a MAC layer that implemnets the asynchronous duty cycling mechanism of described in Chapter 2. Our implementation uses simple packet interfaces, and can be integrated with other link layers such as S-MAC [110] or T-MAC [102] with minimal modifications. The ETPD implementation is part of the lowest part of the MAC and is not cleanly portable. We embed the type of each packet (parity or plain) in the preamble so that it does not require additional bits.

Fig. 8.7 shows a high-level block diagram of the necessary functions. Link-layer acknowledgement and retransmission functions are shown for completeness. Shaded components are those required to support SPaC; white components are also present in a non-packet combining system. On the send path, the inner encoder block implements the plain/parity decision of section 8.3.4. On the receive path, the combiner block performs decoding or merging depending on the types of the input packets. The receive path decision sequence is represented in Fig. 8.8.

Buffer management. Given two received corrupt packets, a node does not know if they correspond to the same original packet, or two different packets. A buffer management strategy is therefore important to reduce the number of attempts to combine two different packets. The current implementation records a timestamp with each received corrupt packet, and discards a corrupt packet after a given timeout, which is determined based on the traffic load and statically configured into the application. Note also that combining is not attempted for packets with differing CRCs (except for multi-hop packets). There are no explicit mechanisms to detect if two corrupt packets come from different originals – whenever the receiver has two corrupt packets, combining is attempted. This means that combining two different packets, which will fail and is a waste of CPU cycles, can sometimes happen.

Our assumption of a low-rate application is key to this design choice: with a short enough timeout, cache pollution and cross-traffic (where two different corrupt

---

[3] This work predates our work on anypath routing; and consequently uses an earlier generation of wireless sensor nodes and different testbeds.

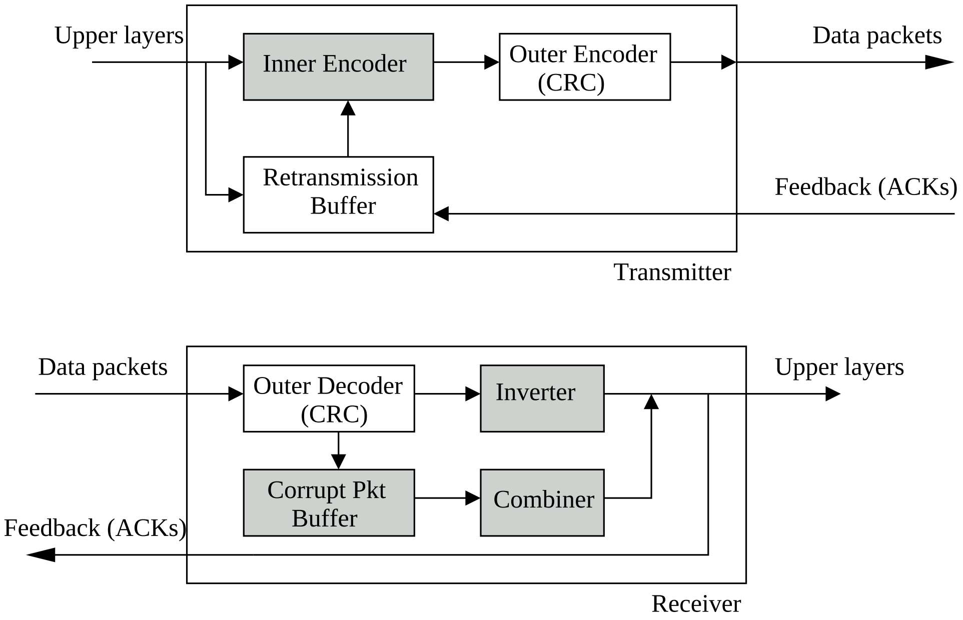


Figure 8.7: Block diagram of packet combining functions on transmit/receive paths.

packets arrive in a smaller interval than the timeout value) are infrequent. More sophisticated buffer management strategies are possible. For example, determining the timeout adaptively, or not buffering a corrupt packet that can only be sent once (such as local broadcast packets sent by a routing protocol). Another option would be to add a randomly chosen identifier (with error protection) to each packet, but the overhead of adding a fixed field to each packet would outweigh the gain from making a few less unnecessary combining attempts.

Multi-hop packets. Routing headers on multi-hop packets usually contain fields that change at each hop, such as next-hop address or distance to destination. Combining packets with differing contents would fail, and so it is necessary to take into account the header fields which may change at each hop. Our current implementation treats multi-hop packets differently from single-hop and broadcast packets, and ignores these fields from the CRC and combining operations. A future improvement will consist of applying the incremental CRC to the modified fields rather than ignoring them.

Memory Footprint. Wherever possible, we used pre-computed table lookups for encoding, decoding and merging. The memory overhead depends on the maximum packet length defined for the application. With the default value of 29 bytes, the ROM footprint of our implementation is 5580 bytes, including code and lookup ta-

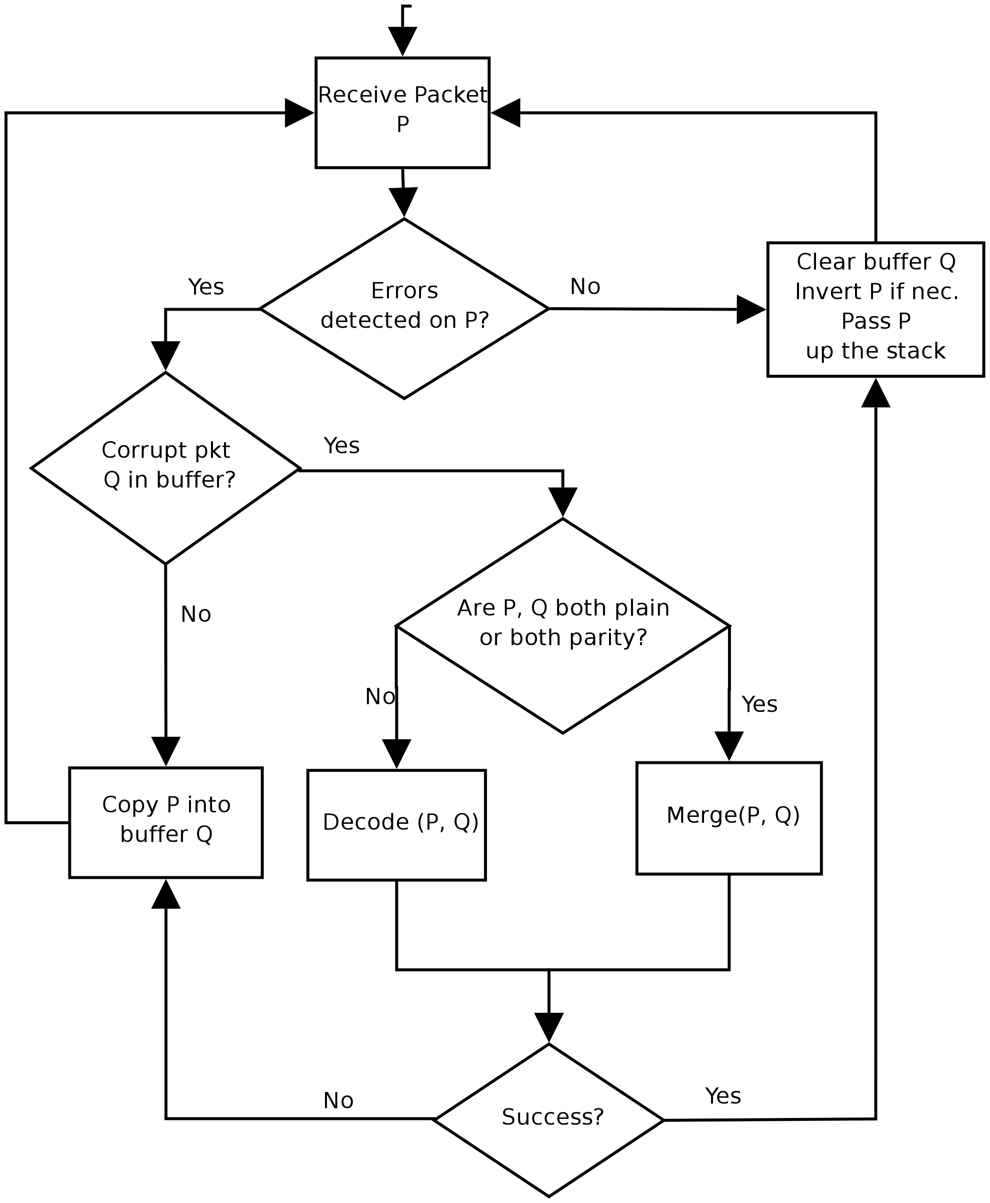


Figure 8.8: Receiver Flowchart.

bles. The RAM footprint includes two packet buffers and totals 158 bytes. A detailed breakdown is given in Table 8.4.

Component	RAM	ROM (code)	ROM (tables)
ETPD	8	450	256
Pkt Combining	$2L + 78$	3674	$16L + 512$

Table 8.4: Memory footprint in bytes, as a function of maximum packet size L.

	CPU Cycles		Equivalent transmitted bits	
Function	L=29 bytes	128	29	128
Encode/Invert	436	1673	1.1	4.3
Decode	2155	9234	5.6	24.1
Diff	1228	4396	3.2	11.4
Merge ($n_e = 2$)	172	172	0.5	0.5
Merge ($n_e = 4$)	1563	1563	4.1	4.1
Merge ($n_e = 6$)	9305	9305	24.2	24.2

Table 8.5: Worst-case CPU overhead of SPaC functions. The CPU energy used for each function is compared with the number of bits that the radio would transmit to use equivalent energy.

8.4.1 CPU Overhead

We evaluated the CPU overhead of all packet combining functions using a combination of manual inspection and scripts that we derived from the PowerTossim [93] distribution. We also converted the computation cost of each function into equivalent communication cost[4] to allow easy comparison between CPU and radio energy costs.

The scripts disassembled the binary image (compiled at `-O3` gcc optimization level) and annotated each line in the `c` sources with the number of corresponding CPU cycles. When a line or function was inlined at different places in the assembly code, with different cycle counts, the script computed the average of the different counts. We then manually summed the total cycles for each function, being careful to account properly for loops (counting the initialization assembly once, and the loop test as many times as the loop is executed).

For most functions, the cycle count depends on the input packets and error patterns. For example, in the case of a packet merging, the procedure exits early if it encounters two candidate error patterns resulting in the same CRC (case $k > 1$, Section 8.3.3), or yet earlier if the merged error mask has more than n_{max} errors. In the case of a packet decoding, the decoding loop exits early if the $(8, 4)$ Hamming code

---

[4] This translation was computed for the case where the radio sends at 0 dBm, using the constants measured in [93]; converting to other transmit powers is simply a matter of multiplying the cycle count by a different constant.

detects an uncorrectable error. We present here the *worst-case* cycle counts for all functions with input-dependent behavior.

Table 8.5 shows worst-case cycles and energy-equivalent radio transmitted bits for each function. Encoding, inverting, and decoding are linear in packet size. Encoding and inverting use negligible CPU energy in comparison to transmission, representing less than 0.5% of the energy to transmit the packet. Decoding is barely more costly in CPU energy, requiring less than 2.5% of a packet transmission cost.

Merging two packets requires first a "diff" operation to compute the merged error mask (Section 8.3.3) by XORing two packets. It is then followed by a search through all candidate error patterns and has cost exponential in n_e. The overhead of merging is sharply higher than that of other functions. We set $n_{max} = 6$ in our experiments, as the worst-case overhead for $n_e > 6$ becomes non-negligible in comparison with a packet transmission cost. For $n_e = 6$, merging is comparable to 7% of the transmission cost for a packet with 29 byte payload, or 2% with a 128 byte payload.

In summary, the energy overhead of SPaC shown by these measurements is far lower than the energy cost of sending a packet, even with the worst-case numbers given in this section. It is nonetheless not negligible, in particular when considering the simplicity of the codes used and the table-driven implementation; computation must therefore be carefully considered in the design of more future, more complex schemes.

8.5 Evaluation

In this section we present performance results for the three networking primitives of Section 8.1 (single-hop with retransmissions, multi-hop routing, and flooding), as well as end-to-end results on a live network. The results in this Section cover the two testbeds of Section 8.2, as well as a 5-node testbed and a semi-permanent deployment, both at EPFL's BC building. Three very different physical environments have therefore been considered; while they cover a substantial ground they are by no means exhaustive.

8.5.1 Single-hop with Retransmissions

We used B-MAC's acknowledgement mechanism with sender-side ACK timeouts increased by 2 radio byte transmission periods to account for the increased receiver processing time (this results in a maximum throughput reduction of less than 4%, in the worst-case where the receiver must merge every two packets with $n_e = 6$). We built a simple retransmission scheme based on the acknowledgement mechanism that

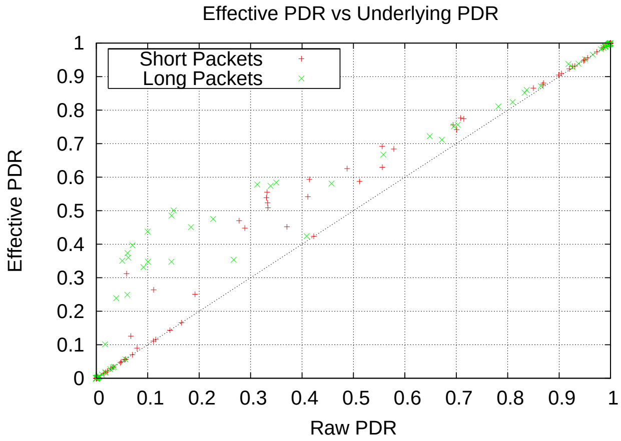
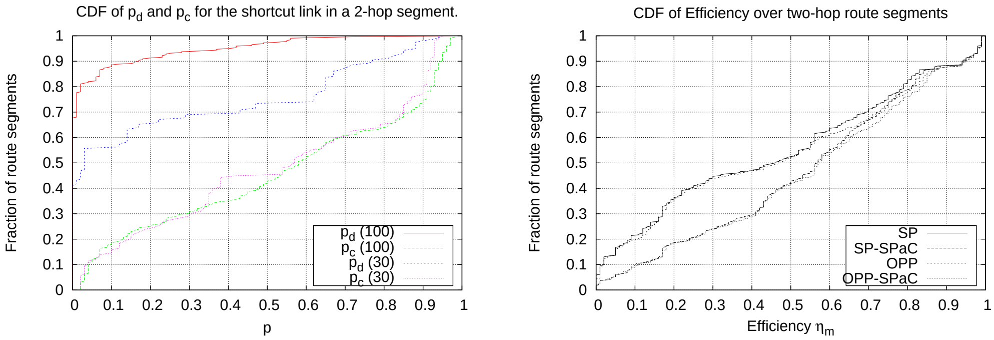


Figure 8.9: Single-hop throughput efficiency η as a function of raw throughput efficiency p_d.

is implemented in B-MAC. When acknowledgements are enabled, a node acknowledges reception of a valid unicast packet addressed to it by transmitting a short (4 byte) "ACK" code. The sender waits a short period for the acknowledgement, and sets `ack` field in the the packet structure appropriately before signalling the `sendDone()` event, allowing the upper layer to take some action (e.g., retransmit the packet) if the packet is unacknowledged.

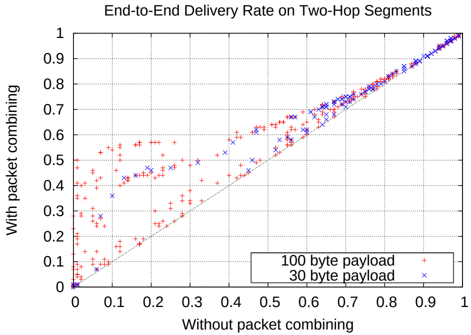
We define the *single-hop throughput efficiency* η (or simply *throughput*) as $1/N_t$, where N_t is the average number of packets transmitted until the packet is successfully accepted at the receiver. The throughput without combining is therefore p_d. The retransmission mechanism retransmits a packet up to a maximum number of attempts T_{max}, until a transmission is positively received. We set $T_{max} = 5$ for the experiments of this section. We ran these experiments over 20 pairwise links (using a 5-node subset of the testbed), with varying transmission power and packet lengths. Nodes dumped all sent and received packets via the serial backchannel, as for our bit-level measurements. When a node successfully combined two corrupt packets into a valid one, this packet was also logged, allowing us to distinguish in the logs which valid packets were the result of a combining operation. In addition, the sender logs showed how many retransmissions were needed for each sent packet.

Fig. 8.9 shows the observed throughput efficiency η as a function of underlying link delivery p_d, as well as the theoretical curve from Appendix 8.7. The theoretical curve qualitatively matches the empirical data, and is a good approximation for $p_d > 0.6$. For lower values of p_d, the theoretical curve has an upward bias. This is due, at least



a) CDF of p_d and p_c for the shortcut link from A to C in a route A, B, C.

b) CDF of efficiency η_m for single-path and anypath routing, with and without SPaC (100-byte packets).



c) Scatter plot comparing end-to-end delivery rate (single-path routing) with and without SPaC.

Figure 8.10: Shortcut link measurements and impact of SPaC on multi-hop routing performance.

in part, to the finite number of retransmissions T_{max} used in the experiments.

The empirical results show that many links with a raw PDR of 5-20% see a throughput increase of over 100%; for some links the increase is more than five-fold. Links with a raw PDR between 30% and 70% see a throughput increase between 5% and 50%. Single-hop combining makes little difference on links above 90% PDR. As expected from the analysis of Section 8.7, the strongest gains come on links with low delivery rate. Single-hop packet combining is therefore most useful on links that are persistently poor, or when links are temporarily degraded by environmental changes.

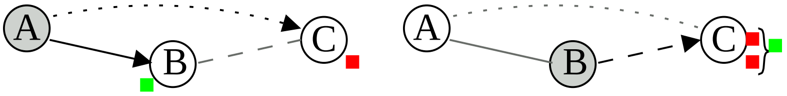


Figure 8.11: Shortcut link on a two-hop segment.

8.5.2 Multi-hop Routing

In multi-hop routing, packet combining can happen using overheard corrupt packets at upstream nodes in the route, in addition to combining hop-by-hop retransmissions. We first consider routing without hop-by-hop retransmissions, in order to see what gains are possible only via multi-hop overhearing. We used trace-driven simulations over traces generated from every node in turn (in a 25-node subset of our testbed) sending a broadcast packet, up to a total of 200 packets per node. Receivers dumped received packets (including corrupt ones) through the backchannel. This was repeated ten times, for three transmission powers, giving us a total of 30 connectivity snapshots. We use Hull et al's metric of *multi-hop efficiency* [42] and define η_m as the number of hops useful packets travel divided by the total number of packet transmissions. Multi-hop efficiency is a key measure because energy is a scarce resource in low-rate sensor networks. We also consider end-to-end reliability of two-hop paths.

For simplicity we consider only the corrupt packets heard one hop ahead of the next hop (i.e., the corrupt packet overheard at node C in Fig. 8.11); the results given here therefore do not incorporate possible gains when a node more than two hops out overhears a packet. We consider a two-hop route segment with three nodes A, B, C as in Fig. 8.11. We note p_d^{AB} and p_d^{BC} the packet delivery rate on the segment's two hops, and p_d^{AC} the delivery rate from A directly to C. Many approaches to link cost estimation and route selection are possible. Due to lack of space, we focus here only on a simple routing metric that maximizes end-to-end delivery rate, with link quality estimation being the average delivery rate seen on that link in the full 200-packet trace. Using the empirical pairwise delivery matrix, we identified all feasible two-hop segments that might be chosen by such a protocol, that is those segments for which $p_d^{AB} p_d^{BC} > p_d^{AC}$. The schemes examined in this section are evaluated over all such feasible segments.

How often the multi-hop form of packet combining from Fig. 8.1 occurs depends on the link between A and C. On a route A, B, C, what quality link exists (if any) between A and C? Does C overhear many packets, good or corrupt, from A? Fig. 8.10(a) shows the CDF of p_d^{AC} and of p_c^{AC}. In many route segments, corrupt packets from A are frequently received at C. For example, in half of the route segments considered, C receives over 40% of A's transmissions (with errors). The number of packets overheard at C without errors is much smaller.

We consider two routing schemes: single-path routing and anypath routing. The difference between both schemes is that in anypath routing, packets from A overheard without errors at C are counted as successful transmissions[5]. Using the packet traces, we then evaluated the efficiency of both schemes with and without packet combining. The results are shown in Fig. 8.10(b) (omitting segments for which $p_d^{AB} \geq 0.99$ and $p_d^{BC} \geq 0.99$, since these already have near-perfect delivery). This data shows that even using only multi-hop combining and no hop-by-hop combining, packet combining often improves the efficiency of two hop route segments. There are two main regions in Fig. 8.10(b). The right half is the region with high efficiency paths. These paths have good links and there is little opportunity for packet combining to improve performance. The left half is the region with low efficiency paths. In this region packet combining is more frequently invoked. With packet combining, less than one-third of paths have $\eta_m < 0.4$, whereas nearly half of paths have $\eta_m < 0.4$ without packet combining. Anypath routing offers a comparatively smaller advantage. Figure 8.10(c) shows the effect of packet combining on end-to-end delivery rates for single-path routing. Since SPaC does not transmit more data than the equivalent link-layer without SPaC, it never decreases performance. As in the case of single-hop with retransmissions, gains are strongest with long packets.

Routing with hop-by-hop retransmissions

We now look at the gains from combining when hop-by-hop retransmissions are used at each hop. The acknowledgement and retransmission mechanism is the same as in Section 8.5.1, with the setting $T_{max} = 3$. Figure 8.12 shows the CDF of multi-hop efficiency η_m for single-path routing, with and without combining. It is qualitatively similar to Fig. 8.10(b), with an overall improvement in efficiency for both protocols. In comparison with Fig. 8.10(b), packet combining offers greater gains with hop-by-hop retransmissions than without. Gains increase with retransmissions because there are more opportunities to combine packets. The third curve in Fig. 8.12 shows efficiency when only hop-by-hop packet combining is allowed, in other words when corrupt transmissions from A overhead at C are not exposed to SPaC. This curve shows that the additional gains from multi-hop combining are appreciable in comparison to the gains from hop-by-hop HARQ.

---

[5] In further work, one could assume a distributed coordination scheme which allows B to learn when C overhears a direct transmission [9]. We do not assume the use of such a scheme, and so B forwards the packet even when C has received it successfully.

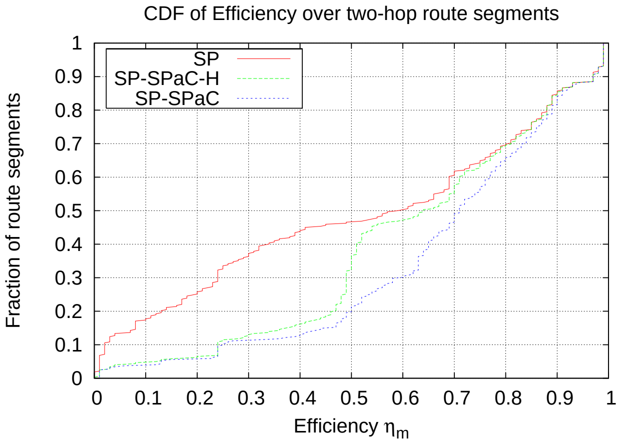
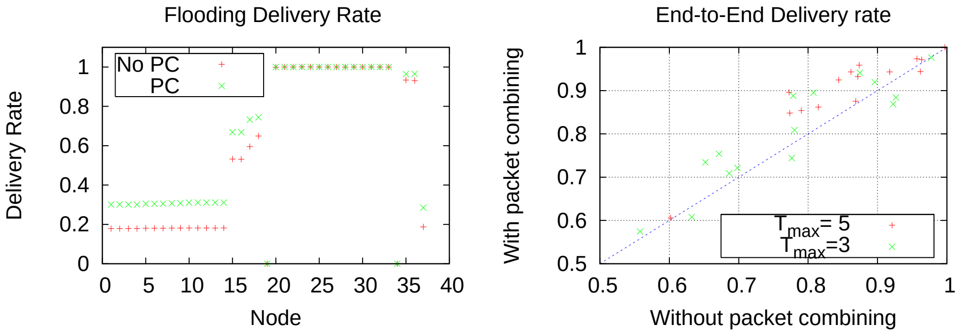


Figure 8.12: CDF of efficiency η_m for single-path routing with retransmissions (SP), with hop-by-hop packet combining only (SP-SPaC-H), and unrestricted packet combining (SP-SPaC). 100-byte packets.

8.5.3 Flooding

We now turn to the network flooding primitive described in Section 8.1.1. The protocol used was the standard TinyOS broadcast implementation (`tos/lib/Broadcast/`), which is a simple flooding protocol with sequence-number based duplicate suppression. We made 6 interleaved runs, with one run consisting of a series of 1000 consecutive floods followed by a second series of 1000 floods with SPaC. The originator was a node in the center of the network, who originated floods at one second intervals. In our first experiments, we observed that with the default backoff timer settings of B-MAC [84], a large number of packets were lost due to collisions. This observation was inferred from the bit-error distribution of received packets: corrupt packets had sharply higher numbers of bit errors than observed in Section 8.2.1, and these bit errors were often present in large bursts. We then increased the MAC layer backoff timers to operate in a non-congested and hence more efficient regime.

Fig. 8.13(a) shows the delivery rate for all 39 nodes, with and without packet combining, with node 22 being the originator. Nodes 19 and 34 were not responding properly and show up as having received no flood packets. A first observation is that packet combining increases the delivery rate for all nodes that did not already have 100% delivery without packet combining. For nodes 1-14, the delivery increased by over 75%, while for nodes 15-18 and 37, delivery increased by over 25%. A second observation is that the graph has marked plateaus (nodes 1-14 and 20-33), within which the increase in delivery rate due to packet combining is constant. These observations



a) b)

Figure 8.13: Flooding delivery rate (left) and scatter plot of end-to-end delivery rate in SensorScope (right).

indicate that the network is heterogeneous, with well connected sub-clusters within which a packet is reliably propagated to every node. Inter-cluster connectivity however is weaker, and it is at the border nodes between clusters that packet combining makes a difference to the overall reception rate. We tested this hypothesis by running similar experiments with different flood originators, and saw that indeed, within either group (nodes 1-14 and 20-33), nodes had similar delivery rates independently of the originator.

The impact of packet combining on flooding clearly depends on the network topology. In a dense (homogeneous) network (and assuming that protocol parameters are properly set to avoid excessive collisions), nodes will receive most flooded packets correctly and packet combining functions are rarely invoked. In a sparse (homogeneous) network with a large number of poor links (such as the example of Figures 8.2), packet combining will operate at many nodes, since they frequently receive more than one corrupt copy of a flooded packet. In the case of a heterogeneous topology such as the one we used, packet combining operates at a small subset of nodes, essentially on either side of the weak links between sub-clusters.

8.5.4 Breakdown of Combining Attempts

The results given above relate the performance improvement from packet combining with the underlying link or path delivery rate. They allow us to compare theoretical and empirical performance, and to evaluate the effect of underlying topologies and channel conditions on the behavior of the scheme.

We now investigate the behavior of packet combining at a more fine-grained level, to gain insight on how frequently merging and combining operations respectively come

	Attempted		**Successful**	
Operation	Single-hop	Multi-Hop	Single-hop	Multi-Hop
Decode	83%	64%	44%	48%
Merge	17%	36%	14%	22%

Table 8.6: Breakdown of attempted and successful combine operations between decode and merging.

into play, and what their associated success rates are.

Table 8.6 gives the breakdown. The left side shows the proportion of attempted merging versus decoding operations. In the single-hop case, merging is attempted less frequently than decoding because the sender alternates between plain and parity packets, and therefore the receiver only has two packets of same type when a packet (or an acknowledgement) is missed and the two adjacent transmissions are corrupt. For multi-hop, the sender randomly chooses between plain and parity for each transmission, and so the breakdown is roughly even.

The second part of Table 8.6 shows the success rate for decoding and merging operations. Overall, combining has a success rate below 40%, underscoring the potential for further gains in systems using more powerful codes or multiple packets. Decoding has a significantly higher success rate than merging. While this difference is explained by the larger number of error patterns that can be corrected with two packets of different types, the overall success rate for merging is still low. We processed our traces to further understand why merging fails so frequently, and found that in a vast majority of cases (over 90%), merging aborts because the number of differing bits between both corrupt packets is greater than n_{max}. Note that in these cases merging aborts at the "diff" step, whose processing requirements are much lower than a full merging (see Table 8.5). Hidden errors account for fewer than 10% of merging failures, suggesting that n_{max} could be increased if more CPU processing power were available, or in a more efficient hardware-based implementation.

8.5.5 End-to-end Performance

We integrated our packet combining implementation into SensorScope [90], an indoor monitoring network deployed at EPFL. SensorScope is a long-running deployment consisting of 18 mica2dot nodes installed throughout a 4 story campus building. The radio stack uses our ETPD implementation, an ACK-based retransmission mechanism, and B-MAC asynchronous duty cycling ([84]). One experiment consisted of logging all sensor data packets received at the basestation over a duration of 40 hours. The network was re-configured at hourly intervals (via broadcast commands) to alternate between enabling and disabling packet combining. This entire experiment was repeated twice, changing the maximum number T_{max} of link layer retransmissions.

We chose SensorScope for our experiment for practical reasons: it was available and under our control. This network should a priori *see little gains from packet combining*, due to three reasons. The first is the network's density: most nodes have high quality paths to the sink most of the time. The second reason is that the network is shallow, with an average sink distance of less than two hops, and the third is that packets are relatively short (19 byte payloads).

Figure 8.13(b) shows the end-to-end data delivery rate for each node, averaged over all 20 runs, with and without packet combining, for two different maximum retransmission values. We see that delivery rate with packet combining is increased for a large majority of nodes, with 30% of the nodes seeing a delivery increase of over 10%. Though such gains are worthwhile given the negligible overhead of packet combining, they are weaker than the microbenchmark gains, due to the unfavorable characteristics of SensorScope discussed above.

The question is then whether SensorScope is representative of all deployed sensor networks. An attempt to answer this question is beyond the scope of this dissertation, and probably premature given that sensor networks are still a nascent field. We simply note in comparison that one of the few well-documented large-scale deployments to date (Great Duck Island [98]) has characteristics making it far more susceptible to packet combining gains: it is larger, has a greater proportion of multi-hop nodes, as well as a larger proportion of weak to intermediate links.

8.6 Summary

This chapter has introduced a novel scheme for error-correction that exploits temporal and spatial diversity through packet combining. Beside hop-by-hop communication, the scheme also works on multi-hop interactions present in routing or broadcasting. As such, it is fundamentally coupled to the broadcast nature of the wireless medium, in contrast to traditional point-to-point FEC techniques that work identically in wired and in wireless networks.

The performance gains of packet combining are promising in light of the extremely simple codes and reduced computation that our system uses. Much room remains to explore systems that employ more powerful codes, and that operate over more than two corrupt packets. Integrating our merging algorithm with the block combining [76] technique of Miu et. al. may lead to improved performance on bursty channels. While a software implementation has low overhead for the short codes used in our scheme, more powerful coding schemes may require hardware support for fast decoding operations.

8.7 Appendix: Single-Hop Analysis

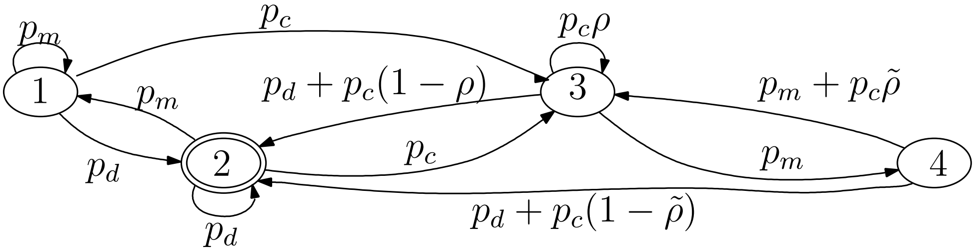


Figure 8.14: Markov chain model for packet combining over a single-hop unicast link. State transitions happen at each packet (re-)transmission.

We analyze the performance gains offered by packet combining as a function of the underlying link quality and of packet length. Our analysis explicitly considers the probability of missed packets, which means that two consecutively received packets may be of same type, even when the sender alternates between plain and parity. We consider a retransmission scheme where the sender repeats the transmission of a packet, alternating between plain and parity encodings, until it has successfully received an acknowledgement from the receiver[6]. A reliable feedback channel is assumed.

We model the sender-receiver pair with a Markov chain (Fig. 8.14) that (self-)transitions at each packet transmission. State 1 is when the receiver has an empty buffer. State 2 is when the receiver has a valid packet, either received directly or as a result of combining two corrupt packets. In states 3 and 4, the receiver has a corrupt packet. In state 3, the sender will next transmit a packet of type opposite to the one in the receiver's buffer; in state 4 the sender will transmit a packet of same type. The Markov chain is irreducible and positive recurrent, and therefore has a unique stationary distribution π.

The receiver successfully receives or combines a packet at every transition spent in state 2. Therefore the throughput efficiency is immediately obtained from the chain's stationary distribution. We omit the the explicit form of π and show here

---

[6] In practice the maximum number of retransmissions is finite; therefore the theoretical throughput considered here is an upper bound on the *achievable* throughput with finite retransmissions.

only $\eta = \pi(2)$:

$$\eta = \frac{(1-p_m)((\tilde{\rho}-1)p_m^2+(\rho-\tilde{\rho}+\tilde{\rho}p_d)\,p_m-\rho+p_d\rho+1)}{(\tilde{\rho}-2)\,p_m^2+(\rho+\tilde{\rho}p_d-p_d-\tilde{\rho})p_m-\rho+p_d\rho-p_d+2} \tag{8.5}$$

Note that for fixed L and L_{pre}, all the quantities in (8.5) are a function of the single parameter p_e (or equivalently, p_d), allowing us to compare the theoretical value of η with empirical results in Fig. 8.9. Substituting $\rho = \tilde{\rho} = 1$ (8.5), gives p_d, the throughput *without* packet combining. In the case when $p_m = 0$, that is, if preambles are always correctly detected, then states 1 and 4 of the Markov chain are never visited, and (8.5) considerably simplifies:

$$\eta = \frac{1 - p_c\rho}{1 + p_c(1-\rho)}. \tag{8.6}$$

We observe that (8.6) does not depend on $\tilde{\rho}$: if $p_m = 0$, the receiver never invokes the merging operation.

Chapter 9

Conclusions

We close this dissertation with an enumeration of some remaining research challenges, some avenues of future work, and a discussion of the outlook for anypath routing.

9.1 Limitations and Open Problems

This dissertation addresses the problem of point-to-point routing in multi-hop wireless networks. Routing is a key component of a networked system's performance, but not the only one, and anypath routing is certainly not a panacea for all the difficulties of building robust and scalable multi-hop wireless systems. Some prior problems that are open with single-path routing remain to be solved for anypath routing. Anypath routing also introduces some new design problems, in particular related to the redesign of upper-layer protocols that are in some way (explicitly or implicitly) predicated on single-path routing. We list some of these open problems here.

Maximizing energy lifetime

Our anycast forwarding methods for asynchronous duty cycling seek to reduce sender energy costs by allowing shorter preambles at a given duty cycle. Note that minimizing a node's overall energy expenditure is not equivalent to minimizing its transmission cost at a given duty cycle. As we have seen, a node consumes energy while listening, and so to maximize lifetime we must *jointly* minimize the duty-cycle and the transmission costs. Therefore, we do not claim to have found the optimal operating regime: our technique decreases energy consumption of nodes relative to single-path routing at the same duty cycle, but it might be possible to reduce consumption further by jointly minimizing transmitter costs and receiver costs.

However, this joint minimization requires knowledge of traffic pattern and of network topology. Furthermore even with single-path routing, techniques to compute the optimal operating points are known only for certain special cases. Still, even if the absolute minimum cannot be found, it remains likely that improved energy performance can be obtained if we consider, in the computation of receiver duty cycles, the expected reduction of preamble lengths due to anycast forwarding.

Congestion

We have assumed from the outset networks with limited congestion, where links are not consistently saturated. Beyond noting that the reduction in transmissions (under ETX) or preamble lengths (under asynchronous duty cycling) will push back the point of congestion, we have neither proposed nor evaluated the operation of anypath routing with high-rate traffic inputs.

Reliability

We have also not tackled the problem of end-to-end reliability. End-to-end reliability is typically implemented as part of a transport layer protocol, and as such operates on top the routing layer. In an ideal layered architecture, we could simply re-use existing reliability schemes with anypath routing; in practice this is rarely the case, as the difficulty of making TCP work effectively over a *single* wireless hop has shown [5]. Note that the reverse path, used as a back-channel from receiver to sender, is not the same with anypath routing as with single-path routing. Options include selecting one path in the anypath and using it as the reverse path (in which case we need a strategy for establishing this path), or using anypath routing for the reverse path itself.

Search Heuristics

For the cost metrics considered in this dissertation, finding optimal candidate relay set fortunately did not require the worst-case search of all $2^{|N(i)|}$ possibilities. Furthermore, even for cost metrics where the simplifications of Section 5.4.2 do not apply, density is unlikely to increase sharply, and thus a brute-force strategy should remain a feasible (if not ideal) approach. However, search heuristics remain to be developed for situations where the number of neighbors is large and where the simplifications of Chapter 5 do not apply.

9.2 Future Directions

In this section we discuss some possible avenues of future work.

Anypath routing in wired networks

As we have shown with synchronous duty cycling, anycast forwarding does not require a broadcast channel in order to be useful. And, as we outlined in Chapter 3, anycast forwarding may be useful in certain wired networks. One such example is a peer-to-peer network where nodes are frequently disconnected; the cost (e.g., delay) to reach *any* node in a group of neighbors is lower than the cost to reach a specific next hop. This remains little more than an intuition, and it remains to be seen in more detail if, and how, anypath routing can be exploited in such a setting.

Another possible application domain is that of *delay tolerant networks* (DTNs), also known as *disruption* tolerant networks [27]. Traditional routing techniques do not apply directly to a DTN because connectivity is intermittent, typically as a result of node mobility. If node mobility patterns are periodic, then each node can has some set of neighbors that each come and go at specific times. Deciding to use a *single* neighbor to reach a destination is suboptimal from the point of view of latency; anypath routing can be used in a manner similar as with synchronous duty cycling in order to reduce path latency. Of course, few mobility patterns are deterministic and periodic; an interesting challenge is to extend anypath routing to DTNs where temporal connectivity patterns have both periodic and random components.

Alternative formulations

Anypath routing is a method for finding paths in communication network. Quite naturally, the language, concepts, and all-around viewpoint used in this dissertation are those of the communications and networking disciplines.

We speculate that there may be a more general formulation of anypath routing, and that such a formulation may allow us to relate anypath routing to other graph-centric algorithmic problems. One more abstract way to formulate anypath routing may be to construct an expanded version of our base graph, where each node i has $2^{|N(i)|}$ neighbors corresponding to each of the candidate relay sets it is “connected” to. The difficulty lies in finding a way to express this graph expansion further than the neighborhood of node i, i.e., defining the neighbors of node i’s neighbors, and the links between them. This appears somewhat reminiscent of the problem of minimum-weight traversals in hypergraphs [55].

Cooperative communication

We believe that the use of anypath routing as an underlying substrate opens up practical roads to explore novel approaches collectively known in theoretical settings as “cooperative communication” [1]. One key aspect of many of these schemes is that they abandon (partially or altogether) the notion of unicast, point-to-point communication, as does anycast forwarding.

Packet combining is one such example of an error-correction mechanism that matches anycast forwarding very naturally. Other approaches are also worth exploring. One example is the integration of network coding with anypath routing. Network coding [52, 56] allows and encourages mixing (i.e., coding) of packets at intermediate network nodes. The use of network coding requires revisiting existing routing notions in order to achieve this mixing of different sources.

Another example is the use of distributed source coding [87] (also known as distributed compression). Distributed source codes allow nodes having correlated data to take advantage of the correlation and each transmit a compressed version of their data, *without* any form of coordination being necessary. If we consider a packet transmission to be correlated data received by multiple candidate relays, and if each relay receives this packet with some errors, then it may be more efficient to for receivers forward compressed versions of these noisy packets using distributed source coding than to retransmit. We would effectively have multiple relays forwarding data from one sender, once again this has similarities with anycast forwarding.

9.3 Discussion and Outlook

An essential feature of anypath routing, in our view, is that all of its concepts and constructs are generalizations of single-path routing. This explains why shortest anypath routes are found using algorithms directly derived from the standard Dijkstra or Bellman-Ford single-path algorithms.

This feature also explains why in the *worst case*, anypath routing behaves identically to single-path routing, either because link costs are not decreased by adding candidates, or because the topology is such that the shortest anypath route is a single path. Outside of this worst case scenario, anypath routing and anycast forwarding jointly result in less costly multi-hop communications than single-path.

Furthermore, the resulting routes are more stable than with other forms of routing, essentially due to the integrative nature of anypath routing cost metrics. This means, in practical terms, that an anypath routing protocol is more robust, must devote less overhead to computing and updating routes, and is more stable with a "noisy" view of network topology than its single-path brethren.

Finally, anypath routing also provides *load balancing*, since packets are spread over different paths. Load balancing is not a primary goal of anypath routing, unlike many multipath routing algorithms; it is in a sense provided "for free" as a side effect of anycast forwarding.

These attributes, in our view, make a strong and broad case for anypath routing. Our choice of wireless sensor networks for the protocol design of Chapter 7 reflects our belief that these are an attractive target: energy longevity is a true challenge in many applications, and furthermore the difficult and expensive step of relay arbitration can be avoided in the context of low-power, low-rate link layers. Of course the true measure of success for any networking protocol lies in its adoption and long-term use. We have at this point laid the foundation of a promising new approach, but only time will allow us to see whether and how this approach matures into a set of enduring and operational network protocols.

9.4 Summary

In this dissertation, we have shown that anycast forwarding is a general and practical way to exploit spatial diversity in multi-point wireless communication. In order to take full advantage of anycast forwarding, we have had to revisit the notion of a path, its associated metrics, and the protocols used to compute these paths. This results in a generalization of single-path routing that we call *anypath routing*. We have also designed, implemented, and evaluated a novel scheme for error-correction that exploits temporal and spatial diversity through packet combining.

Bibliography

[1] T.E. Hunter A. Nosratinia and A. Hedayat. Cooperative Communication in Wireless Networks. *IEEE Communications Magazine*, 42(10):68–73, Oct 2004.

[2] Robert Adler, Phil Buonadonna, Jasmeet Chhabra, Mick Flanigan, Lakshman Krishnamurthy, Nandakishore Kushalnagar, Lama Nachman, and Mark Yarvis. Design and Deployment of Industrial Sensor Networks: Experiences from the North Sea and a Semiconductor Plant. In *Proceedings of ACM Sensys*, San Diego, USA, November 2005.

[3] Alec Woo and David Culler. Evaluation of Efficient Link Reliability Estimators for Low-Power Wireless Networks. Technical report, UCB, November 2003.

[4] Muneeb Ali, Umar Saif, Adam Dunkels, Thiemo Voigt, Kay Roemer, Koen Langendoen, Joseph Polastre, and Zartash Afzal Uzmi. Medium Access Control Issues in Sensor Networks. *SIGCOMM Comput. Commun. Rev.*, 36(2):33–36, 2006.

[5] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.

[6] Richard Bellman. On a Routing Problem. *Quarterly Journal of Applied Mathematics*, 16(1):87–90, 1958.

[7] Dimitri Bertsekas and Robert Gallager. *Data Networks; second edition.* Prentice Hall, 1992.

[8] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang. MACAW: A Media Access Protocol for Wireless LAN's. In *SIGCOMM '94: Proceedings of the Conference on Communications Architectures, Protocols and Applications*, pages 212–225, New York, NY, USA, 1994. ACM Press.

[9] Sanjit Biswas and Robert Morris. Opportunistic Routing in Multi-Hop Wireless Networks. In *Proceedings of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM)*, Philadelphia, USA, 2005.

[10] Richard E. Blahut. *Theory and Practice of Error Control Coding.* Addison-Wesley, 1983.

[11] Florian Braun and Marcel Waldvogel. Fast Incremental CRC Updates for IP over ATM networks. In *2001 IEEE Workshop on High Performance Switching and Routing (HPSR 2001)*, pages 48–52, Dallas, TX, USA, May 2001.

[12] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Mobile Computing and Networking*, pages 85–97, 1998.

[13] Alberto Cerpa, Naim Busek, and Deborah Estrin. SCALE: A Tool for Simple Connectivity Assessment in Lossy Environments. In *CENS Technical Report 0021*, 2003.

[14] Chipcon. CC1000 Transceiver Datasheet. http://www.chipcon.com.

[15] Romit Roy Choudhury and Nitin H. Vaidya. MAC-layer Anycasting in Ad Hoc Networks. *SIGCOMM Comput. Commun. Rev.*, 34(1):75–80, 2004.

[16] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit Disk Graphs. *Discrete Math.*, 86(1-3):165–177, 1990.

[17] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom '03)*, San Diego, USA, 2003.

[18] David Walden. The Bellman-Ford Algorithm and "Distributed Bellman-Ford". http://www.walden-family.com/public/bf-history.pdf.

[19] Luca de Alfaro and Angelo Raffaele Meo. Codes for Second and Third Order GH-ARQ Schemes. *IEEE Trans. on Communications*, 1994.

[20] S. N. Diggavi, N. A. Dhair, A. Stamoulis, and A. R. Calderbank. Great expectations: The value of spatial diversity in wireless networks. *Proceedings of the IEEE*, 92:219–270, February 2004.

[21] Edsger. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. In *Numerische Mathematik*, volume 1, pages 269–271. Mathematisch Centrum, Amsterdam, The Netherlands, 1959.

[22] Olivier Dousse. Asymptotic Properties of Wireless Multi-Hop Networks. In *PhD Thesis, EPFL, no 3310*, 2005.

[23] Henri Dubois-Ferrière, Roger Meier, Laurent Fabre, and Pierre Metrailler. Tinynode: A Comprehensive Platform for Wireless Sensor network Applications. In *Fifth International Conference on Information Processing in Sensor Networks (IPSN)*, Nashville, Tenessee, April 2006.

[24] Deborah A. Dyson and Zygmunt J. Haas. A Dynamic Packet Reservation Multiple Access Scheme for Wireless ATM. *Mob. Netw. Appl.*, 4(2):87–99, 1999.

[25] A. El-Hoiydi, J. Decotignie, and J. Hernandez. Low Power MAC Protocols for Infrastructure Wireless Sensor Networks, 2003.

[26] Amre El-Hoiydi. Aloha with Preamble Sampling for Sporadic Traffic in Ad Hoc Wireless Sensor Networks. In *Proceedings of IEEE International Conference on Communications*, April 2002.

[27] Kevin Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34, New York, NY, USA, 2003. ACM Press.

[28] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. *IEEE/ACM Trans. Netw.*, 5(6):784–803, 1997.

[29] R. Fonseca, S. Ratnasamy, D. Culler, S. Shenker, and I. Stoica. Beacon Vector Routing: Scalable Point-to-point Routing in Wireless Sensornets. In *Proceedings of 2nd Symposium on Networked Systems Design and Implementation (NSDI '05)*, 2005.

[30] Saurabh Ganeriwal, Deepak Ganesan, Hohyun Shim, Vlasios Tsiatsis, and Mani B. Srivastava. Estimating Clock Uncertainty for Efficient Duty-Cycling in Sensor Networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 130–141, New York, NY, USA, 2005. ACM Press.

[31] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 251–254, New York, NY, USA, 2001. ACM Press.

[32] Deepak Ganesan, Bhaskar Krishnamachari, Alec Woo, David Culler, Deborah Estrin, and Stephen Wicker. Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks. In *UCLA Computer Science Technical Report UCLA/CSD-TR 02-0013*, 2003.

[33] Andrea Goldsmith. *Wireless Communications.* Cambridge University Press, New York, NY, USA, 2005.

[34] Jonathan Goodman, Albert G. Greenberg, Neal Madras, and Peter March. Stability of Binary Exponential Backoff. *J. ACM*, 35(3):579–602, 1988.

[35] A.G Greenberg and B. Hajek. Deflection Routing in Hypercube Networks. *IEEE Transactions on Communications*, June 1992.

[36] Matthias Grossglauser and Martin Vetterli. Locating Mobile Nodes with EASE: Learning Efficient Routes from Encounter Histories Alone. *IEEE Transactions on Networking*, June 2006.

[37] V. Handziski, J. Polastre, J. H. Hauer, and C. Sharp. Flexible hardware abstraction of the TI MSP430 microcontroller in TinyOS. In *Proc. of SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 277–278, Baltimore, MD, USA, November 2004. ACM Press.

[38] Herbert A. David and H. N. Nagaraja. *Order Statistics.* Wiley, 2003. Third Edition.

[39] Wen Hu, Van Nghia Tran, Nirupama Bulusu, Chun Tung Chou, Sanjay Jha, and Andrew Taylor. The Design and Evaluation of a Hybrid Sensor Network for Cane-toad Monitoring. In *Proceedings of Information Processing in Sensor Networks (IPSN 2005/SPOTS 2005)*, Los Angeles, April 2005.

[40] Jonathan W. Hui and David Culler. The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94. ACM Press, 2004.

[41] C. Huitema. An Anycast Prefix for 6to4 Relay Routers. IETF, RFC, June 2001.

[42] Bret Hull, Kyle Jamieson, and Hari Balakrishnan. Mitigating Congestion in Wireless Sensor Networks. In *ACM SenSys 2004*, Baltimore, MD, November 2004.

[43] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed Diffusion: a Scalable and Robust Communication Paradigm for Sensor Networks. In *Mobile Computing and Networking*, pages 56–67, 2000.

[44] Digi International. Digi Connect ME and Digi Connect Wi-ME. http://www.digi.com.

[45] Shweta Jain and Samir R. Das. Exploiting Path Diversity in the Link Layer in Wireless Ad Hoc Networks. In *WOWMOM '05: Proceedings of the Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM'05)*, pages 22–30, Washington, DC, USA, 2005. IEEE Computer Society.

[46] D. Johnson, D. Maltz, and J. Broch. The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. In *Ad Hoc Networking*. Addison-Wesley, 2001.

[47] L.R. Ford Jr. Network Flow Theory. Paper P-923, The RAND Corporation, Santa Monica, California, August 1956.

[48] J. Jubin and J. Tornow. The DARPA Packet Radio Network Protocols. *Proceedings of the IEEE*, 75(1): 21-32, 1987.

[49] P. Karn. MACA: A new Channel Access Method for Packet radio. In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, 1990.

[50] Mark J. Karol, Kai Y. Eng, and Zhao Liu. An Efficient Demand-Assignment Multiple Access Protocol for Wireless Packet (ATM) Networks. *Wirel. Netw.*, 1(3):269–279, 1995.

[51] Brad Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, New York, NY, USA, 2000. ACM Press.

[52] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Medard, and Jon Crowcroft. XORs in The Air: Practical Wireless Network Coding. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, 2006.

[53] A. Khanna and J. Zinky. The Revised ARPANET Routing Metric. *SIGCOMM Comput. Commun. Rev.*, 19(4):45–56, 1989.

[54] D. Kim, D. Meyer, H. Kilmer, and D. Farinacci. Anycast Rendevous Point (RP) mechanism using Protocol Independent Multicast (PIM) and Multicast Source Discovery Protocol (MSDP). IETF, RFC, 2003.

[55] Donald E. Knuth. A Generalization of Dijkstra's Algorithm. *Information Processing Letters*, 1:1–5, 1977.

[56] Ralf Koetter and Muriel Médard. An Algebraic Approach to Network Coding. *IEEE/ACM Trans. Netw.*, 11(5):782–795, 2003.

[57] A. Köpke. Uses of Channel Codes and Checksums to Improve Energy Efficiency in Sensor Networks. Technical Report TKN-03-008, Telecommunication Networks Group, Technische Universität Berlin, May 2003.

[58] Fabian Kuhn, Roger Wattenhofer, Yan Zhang, and Aaron Zollinger. Geometric Ad-Hoc Routing: Of Theory and Practice. In *22nd ACM Symposium on the Principles of Distributed Computing (PODC), Boston, Massachusetts, USA*, July 2003.

[59] J. N. Laneman, D. N. C. Tse, and G. W. Wornell. Cooperative Diversity in Wireless Networks: Efficient Protocols and Outage Behavior. *IEEE Trans. on Information Theory*, 50, 2004.

[60] Koen Langendoen and Gertjan Halkes. Energy-efficient medium access control. In *Embedded Systems Handbook*. CRC Press, 2005.

[61] Peter Larsson. Selection Diversity Forwarding in a Multihop Packet Radio Network with Fading Channel and Capture. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(4):47–54, 2001.

[62] Sung-Ju Lee, Elizabeth M. Belding-Royer, and Charles E. Perkins. Scalability Study of the Ad Hoc On-Demand Distance Vector Routing Protocol. In *International Journal on Network Management (IJNM)*, 2003.

[63] Will E. Leland, Murad S. Taqq, Walter Willinger, and Daniel V. Wilson. On the Self-Similar Nature of Ethernet Traffic. In Deepinder P. Sidhu, editor, *ACM SIGCOMM*, pages 183–193, San Francisco, California, 1993.

[64] Philip Levis, David Gay, Vlado Handziski, Jan-Hinrich. Hauer, Ben Greenstein, Martin Turon, Jonathan Hui, Kevin Klues, Cory Sharp, Robert Szewczyk, Joe Polastre, Philip Buonadonna, Lama Nachman, Gilman Tolle, David Culler, and Adam Wolisz. T2: A Second Generation OS For Embedded Sensor Networks. Number TKN-05-007, November 2005.

[65] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Network. In *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004)*, 2004.

[66] Yuan Li, Wei Ye, and John Heidemann. Energy and Latency Control in Low Duty Cycle MAC Protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, New Orleans, LA, USA, March 2005.

[67] Shu Lin and Daniel J. Costello. *Error Control Coding, Second Edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.

[68] R.R. Rao M. Zorzi. Geographic Random Forwarding (GeRaF) for Ad Hoc and Sensor Networks: Energy and Latency Performance. *IEEE Trans. on Mobile Computing*, 2, 2003.

[69] Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong. TinyDB: An Acqusitional Query Processing System for Sensor Networks. *ACM TODS*, 2005.

[70] M. Marina and S. Das. On-Demand Multipath Distance Vector Routing in Ad Hoc Networks (AOMDV), 2001.

[71] Nicholas Maxemchuk. Dispersity Routing. In *Proceedings of ICC*, San Francisco, CA, 1975.

[72] Nicholas Maxemchuk. Dispersity Routing In Store-And-Forward Networks. In *PhD Thesis, University of Pennsylvania*, 1975.

[73] Nicholas F. Maxemchuk. Dispersity Routing on ATM Networks. In *INFOCOM*, pages 347–357, 1993.

[74] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven Layered Multicast. In *ACM SIGCOMM*, volume 26,4, pages 117–130, New York, August 1996. ACM Press.

[75] J. M. McQuillan, G. Falk, and I. Richer. A Review of the Development and Performance of the ARPANET Routing Algorithm. *IEEE Transactions on Communications*, pages 1802–1811, 1978.

[76] Allen K. Miu, Hari Balakrishnan, and Can E. Koksal. Improving Loss Resilience with Multi-Radio Diversity in Wireless Networks. In *11th ACM MOBICOM Conference*, Cologne, Germany, September 2005.

[77] Michel Mouly and Marie-Bernadette Pautet. *The GSM System for Mobile Communications*. Telecom Publishing, 1992.

[78] The Institute of Electrical and Electronics Engineers. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Computer Society, October 1999.

[79] The Institute of Electrical and Electronics Engineers. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). IEEE Computer Society, October 2003.

[80] Tom Parker and Koen Langendoen. Guesswork: Robust Routing in an Uncertain World. In *2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, November 2005.

[81] C. Partridge, T. Mendex, and W. Milliken. Host Anycasting Service. IETF, RFC, 1993.

[82] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir R. Das. Ad Hoc On Demand Distance Vector (AODV) Routing. IETF, RFC, July 2003.

[83] Sridhar Pingali, Don Towsley, and James F. Kurose. A Comparison of Sender-initiated and Receiver-initiated Reliable Multicast Protocols. In *SIGMETRICS '94: Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and Modeling of Computer Systems*, pages 221–230, New York, NY, USA, 1994. ACM Press.

[84] Joe Polastre, Jason Hill, and David Culler. Versatile Low Power Media Access for Wireless Sensor networks. In *Proceedings of ACM Sensys*, Los Angeles, USA, April 2003.

[85] Joseph Polastre, Jonathan Hui, Philip Levis, Jerry Zhao, David Culler, Scott Shenker, and Ion Stoica. A Unifying Link Abstraction for Wireless Sensor Networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 76–89, New York, NY, USA, 2005. ACM Press.

[86] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *The Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, Los Angeles, California, April 2005.

[87] S. Sandeep Pradhan and Kannan Ramchandran. Distributed source coding using syndromes (discus): Design and construction. *IEEE Transactions on Information Theory*, 49(3):626–643, 2003.

[88] Venkatesh Rajendran, Katia Obraczka, and J. J. Garcia-Luna-Aceves. Energy-efficient Collision-free Medium Access Control for Wireless Sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 181–192, New York, NY, USA, 2003. ACM Press.

[89] Ananth Rao, Christos Papadimitriou, Scott Shenker, and Ion Stoica. Geographic Routing without Location Information. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 96–108, New York, NY, USA, 2003. ACM Press.

[90] Thomas Schmid, Henri Dubois-Ferrière, and Martin Vetterli. SensorScope: Experiences with a Wireless Building Monitoring Sensor Network. In *Workshop on Real-World Wireless Sensor Networks (REALWSN'05)*, 2005.

[91] Rahul C. Shah, Sven Wietholter, Adam Wolisz, and Jan M. Rabaey. Modeling and Analysis of Opportunistic Routing in Low Traffic Scenarios. In *WIOPT '05: Proceedings of the Third International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pages 294–304, Washington, DC, USA, 2005. IEEE Computer Society.

[92] Rahul C. Shah, Sven Wiethölter, Adam Wolisz, and Jan M. Rabaey. When Does Opportunistic Routing Make Sense? In *3rd IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2005 Workshops), 8-12 March 2005, Kauai Island, HI, USA*, pages 350–356. IEEE Computer Society, 2005.

[93] Victor Shnayder, Mark Hempstead, Bor rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the Power Consumption of Large-Scale Sensor Network Applications. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 188–200, New York, NY, USA, 2004. ACM Press.

[94] P. Sindhu. Retransmission Error Control with Memory. *IEEE Transactions on Communications*, pages 473–479, 1977.

[95] Krishna M. Sivalingam, Jyh-Cheng Chen, Prathima Agrawal, and Mani B. Srivastava. Design and Analysis of Low-power Access Protocols for Wireless and Mobile ATM Networks. *Wirel. Netw.*, 6(1):73–87, 2000.

[96] Anand Srinivas and Eytan Modiano. Minimum Energy Disjoint Path Routing in Wireless Ad-hoc Networks. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 122–133, New York, NY, USA, 2003. ACM Press.

[97] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a Sensor Network Expedition. In *1st European Workshop on Wireless Sensor Networks*, 2003.

[98] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, and David Culler. An Analysis of a Large Scale Habitat Monitoring Application. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Baltimore, November 2004.

[99] D. Thaler and C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection. IETF, RFC, November 2000.

[100] Gilman Tolle, Joseph Polastre, Robert Szewczyk, Neil Turner, Kevin Tu, Phil Buonadonna, Stephen Burgess, David Gay, Wei Hong, Todd Dawson, and David Culler. A Macroscope in the Redwoods. In *Proceedings of ACM Sensys*, San Diego, USA, November 2005.

[101] David Tse and Pramod Viswanath. *Fundamentals of Wireless Communication*. Cambridge University Press, New York, NY, USA, 2005.

[102] Tijs van Dam and Koen Langendoen. An Adaptive energy-efficient MAC Protocol for Wireless Sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180, New York, NY, USA, 2003. ACM Press.

[103] V.Handziski, J. Polastre, J.-H. Hauer, C. Sharp, A. Wolisz, and D. Culler. Flexible Hardware Abstraction for Wireless Sensor Networks. In *Proc. of 2nd European Workshop on Wireless Sensor Networks (EWSN 2005)*, Istanbul, Turkey, February 2005.

[104] Scott Weber and Liang Cheng. A Survey of Anycast in IPv6 Networks. *IEEE Communications Magazine*, 42(1):127–132, January 2004.

[105] Wei Ye and John Heidemann. Ultra-Low Duty Cycle MAC with Scheduled Channel Polling. Technical Report ISI-TR-2005-604b, USC/Information Sciences Institute, July 2005.

[106] Geoffrey Werner-Allen, Geetika Tewari, Ankit Patel, Matt Welsh, and Radhika Nagpal. Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects. In *SenSys '05: Proceedings of the 3rd International conference on Embedded Networked Sensor Systems*, pages 142–153, New York, NY, USA, 2005. ACM Press.

[107] Kamin Whitehouse, Gilman Tolle, Jay Taneja, Cory Sharp, Sukun Kim, Jaein Jeong, Jonathan Hui, Prabal Dutta, and David Culler. Marionette: using RPC for Interactive Development and Debugging of Wireless Embedded Networks. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 416–423, New York, NY, USA, 2006. ACM Press.

[108] Alec Woo, Terence Tong, and David Culler. Taming the Underlying Issues for Reliable Multihop Routing in Sensor Networks. In *Proceedings of ACM Sensys*, Los Angeles, USA, April 2003.

[109] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A Wireless Sensor Network for Structural Monitoring. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Baltimore, November 2004.

[110] Wei Ye, John Heidemann, and Deborah Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the IEEE Infocom*, pages 1567–1576, New York, NY, USA, June 2002. USC/Information Sciences Institute, IEEE.

[111] Y. Yu, B. Krishnamachari, and V. Prasanna. Energy-Latency Tradeoffs for Data Gathering in Wireless Sensor Networks, 2004.

[112] William T. Zaumen and J. J. Garcia-Luna-Aceves. Loop-Free Multipath Routing Using Generalized Diffusing Computations. In *INFOCOM (3)*, pages 1408–1417, 1998.

[113] Bin Zhao and Matthew C. Valenti. Practical Relay Networks: A Generalization Of Hybrid-ARQ. *IEEE Journal on Selected Areas in Communications (Special Issue on Wireless Ad Hoc Networks)*, 23, 2005.

[114] Jerry Zhao and Ramesh Govindan. Understanding Packet Delivery Performance in Dense Wireless Sensor Networks. In *Proceedings of ACM Sensys*, Los Angeles, USA, April 2003.

[115] Z. Zhong, J. Wang, G.-H. Lu, and S. Nelakuditi. On Selection of Candidates for Opportunistic AnyPath Forwarding *(Extended abstract)*. In *Poster Session of ACM MOBICOM'05*, August 2005.

[116] Gang Zhou, Tian He, Sudha Krishnamurthy, and John A. Stankovic. Impact of Radio Irregularity on Wireless Sensor Networks. In *Proceedings of the Second International Conference on Mobile Systems, Applications, and Services (MobiSys 2004)*, Boston, MA, 2004.

[117] Aaron Zollinger. Networking Unleashed: Geographic Routing and Topology Control in Ad Hoc and Sensor Networks. In *PhD Thesis, ETH Zurich, Diss. ETH No. 16025*, September 2005.

[118] Marco Zuniga and Bhaskar Krishnamachari. Analyzing the Transitional Region in Low Power Wireless Links. In *First IEEE International Conference on Sensor and Ad hoc Communications and Networks (SECON)*, Santa Clara, CA, USA, 2004.

Curriculum Vitae

Henri Dubois-Ferrière

1, rue Mathurin-Cordier
1005 Lausanne
Switzerland

Education

2002 - 2006	**Ph.D.** in Computer and Communication Sciences, Audiovisual Communications Laboratory (LCAV), School of Computer and Communication Sciences, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland.
1996 - 2001	**M.Sc.** in Communication Systems, School of Computer and Communication Sciences, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland.
1997 - 1998	**Visiting student**, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. Accepted with scholarship on competitive basis of academic results.

Professional Experience

2002 - 2006	**Research and teaching assistant**, Audiovisual Communications Laboratory (LCAV), I&C, EPFL, Switzerland.
Summers 2004-2005	**Visiting scholar**, Center for Embedded Networked Sensing, UCLA.
2000 - 2002	**Software engineer**, FastForward Networks/Inktomi, San Francisco, CA.

Summer 1999 **Intern**, LCAV/EPFL and Dartfish, Lausanne, Switzerland.

Summer 1998 **Intern**, High-performance Networking Group, SGI, Mountain View, CA.

Publications

1. H. Dubois-Ferrière, M. Grossglauser, and M. Vetterli. **Least-Cost Opportunistic Routing.** *EPFL Technical Report LCAV-REPORT-2007-001 (in submission)*, March 2007
2. H. Dubois-Ferrière, R. Meier, L. Fabre, and P. Metrailler. **TinyNode: A Comprehensive Platform for Wireless Sensor Networking Applications.** *In Proc. 5th International Conference on Information Processing in Sensor Networks (IPSN)*, April 2006.
3. H. Dubois-Ferrière, D. Estrin, and M. Vetterli. **Packet Combining in Sensor Networks.** In *Proc. 3rd ACM Conference on Embedded Networked Sensor Systems (Sensys)* November 2005.
4. T. Schmid, H. Dubois-Ferrière, and M. Vetterli. **SensorScope: Experiences with a Wireless Building Monitoring Sensor Network.** In *Proc. First Workshop on Real-World Wireless Sensor Networks (REALWSN'05)*, June 2005.
5. H. Dubois-Ferrière, D. Estrin, and T. Stathopoulos. **Efficient and Practical Query Scoping in Sensor Networks.** In *Proc. IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, October 2004.
6. H. Dubois-Ferrière, M. Grossglauser, and M. Vetterli. **Space-Time Routing in Ad Hoc Networks.** In *Proc. 2nd International Conference on AD-HOC Networks and Wireless* (also in *Lecture Notes in Computer Science, Nr. 2865*), October 2003.
7. H. Dubois-Ferrière, M. Grossglauser, and M. Vetterli. **Age Matters: Efficient Route Discovery in Mobile Ad Hoc Networks Using Encounter Ages.** *In Proc ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, June 2003.