The SoftPHY Abstraction: from Packets to Symbols in Wireless Network Design

by

Kyle Andrew Jamieson

S.B., Mathematics (2000); S.B. Computer Science and Engineering (2001);
M.Eng., Electrical Engineering and Computer Science (2002)
Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

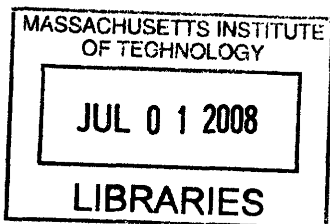
June 2008

© Massachusetts Institute of Technology 2008. All rights reserved.

Author ....
Department of Electrical Engineering and Computer Science
May 23, 2008

Certified by ....
Hari Balakrishnan
Professor
Thesis Supervisor

Accepted by ....
Terry P. Orlando
Chair, Department Committee on Graduate Students



ARCHIVES

The SoftPHY Abstraction: from Packets to Symbols in Wireless Network Design

by

Kyle Andrew Jamieson

Submitted to the Department of Electrical Engineering and Computer Science on May 23, 2008, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science and Engineering

ABSTRACT

At ever-increasing rates, we are using wireless systems to communicate with others and retrieve content of interest to us. Current wireless technologies such as WiFi or Zigbee use forward error correction to drive bit error rates down when there are few interfering transmissions. However, as more of us use wireless networks to retrieve increasingly rich content, interference increases in unpredictable ways. This results in errored bits, degraded throughput, and eventually, an unusable network. We observe that this is the result of higher layers working at the packet granularity, whereas they would benefit from a shift in perspective from whole packets to individual symbols.

From real-world experiments on a 31-node testbed of Zigbee and software-defined radios, we find that often, not all of the bits in corrupted packets share fate. Thus, today's wireless protocols retransmit packets where only a small number of the constituent bits in a packet are in error, wasting network resources. In this dissertation, we will describe a physical layer that passes information about its confidence in each decoded symbol up to higher layers. These *SoftPHY* hints have many applications, one of which, more efficient link-layer retransmissions, we will describe in detail. *PP-ARQ* is a link-layer reliable retransmission protocol that allows a receiver to compactly encode a request for retransmission of only the bits in a packet that are likely in error. Our experimental results show that PP-ARQ increases aggregate network throughput by a factor of approximately 2× under various conditions. Finally, we will place our contributions in the context of related work and discuss other uses of SoftPHY throughout the wireless networking stack.

Thesis supervisor: Hari Balakrishnan
Title: Professor

ACKNOWLEDGMENTS

My advisor Hari Balakrishnan is the individual who has impacted my thinking about wireless networks and research in systems and networking the most, and his influence is found throughout this work. I thank him for the intellectual dialog we have had over the years he has been my advisor. Conversations with Hari tend to be full of excitement and technical insight, and I will look back fondly on the past seven years I have spent in the Networks and Mobile Systems (NMS) research group he leads. I thank him for his support and encouragement in so many ways over this time, without which this dissertation would not have been written.

I came to know thesis committee member Andrew Campbell through our common interest in sensor networking congestion control protocols and medium access control protocols. I thank him for his kind encouragement, deep insights, and constructive criticisms throughout this process, and for traveling to attend my defense.

I have benefited from many conversations with thesis committee member Robert Morris over the years. In a class he taught, his half-hour dissection of one data plot inspired me to think carefully about systems performance data when I examine it. I thank him for his infectious personality and deep insights into all aspects of the research process.

I have had the privilege of knowing and admiring the work of thesis committee member Dina Katabi since the beginning of my graduate career at MIT when I was an undergraduate UROP. I thank her for her participation in my defense.

I thank John Guttag for good advice and good taste.

Barbara Liskov gave me support and encouragement at the beginning of my graduate program at MIT, for which I am deeply grateful.

Patrick Winston gives a yearly lecture titled "How to Speak," which helped me when I needed it most as I prepared my job talk.

To the present, I retain my undergraduate observations of Dennis Freeman's teaching as a goal and reference for my own teaching.

I am grateful to the support staff and staff in The Infrastructure Group (t!g) at the MIT Computer Science and Artificial Intelligence Laboratory and the former Laboratory of Computer Science. In particular, Ty Sealy and Tom Buehler respectively enabled the projection and video recording of my defense seminar.

Sheila Marian has held life in the NMS group together in more ways than one, keeping everything running smoothly. I thank her for this and for sharing the products of her excellent baking skills with the group.

I thank the National Science Foundation for funding much of my graduate studies.

I thank the following people whom I have been fortunate enough to meet during my time at MIT:

Daniel Abadi Atul Adya Dan Aguayo
John Anckorn David Andersen Arvind
Magdalena Balazinska Manish Bhardwaj
Ken Barr Wes Beebee John Bicket
Sanjit Biswas Chuck Blake Chandra Boyapati
Micah Brodsky Vladimir Bychkovsky
Jennifer Carlisle Miguel Castro Wesley Chan
Benjie Chen James Cowling Russ Cox
Dorothy Curtis Frank Dabek Douglas De Couto
Waseem Daher Srini Devadas Jakob Eriksson
Nick Feamster Laura Feeney Miguel Ferreira
Janet Fischer Bryan Ford Thomer Gil
Lewis Girod Shyamnath Gollakota
Michel Goraczko Ramki Gummadi
Joanne Talbot Hanley Jamey Hicks
Anne Hunter Bret Hull Jaeyeon Jung
Frans Kaashoek Srikanth Kandula Sachin Katti
Maxwell Krohn Nate Kushman Butler Lampson
Chris Lesniewski-Laas Jinyang Li
Jenny Liu Neena Lyall Nancy Lynch
Sheila Marian Sam Madden Umberto Malesci
Mary McDavitt Silvio Micali Paula Mickevich
Allen Miu Athicha Multhitacharoen
Daniel Myers Ryan Newton Mike Perrott
Mujde Pamuk Max Poletto Raluca Popa
Bodhi Priyantha Dr. and Ms. Laura and M. Q Qin
Asfandyar Qureshi Hariharan Rahul Sean Rhea
Martin Rinard Stanislav Rost Larry Rudolph
Jerry Saltzer Jeff Sheldon Eugene Shih
Ali Shoeb Liuba Shrira Emil Sit
Alex Snoeren Charlie Sodini Jeremy Stribling
Zeeshan Syed Jayashree Subramanian
Godfrey Tan Chon-Chon Tang
Chris Terman Arvind Thiagarajan

Sivan Toledo   Eduardo Torres-Jara
Ben Vandiver   Mythili Vutukuru   Keith Winstein
Grace Woo   Alex Yip   Yang Zhang

I have had the privilege of meeting the following people outside the Institute, and I thank them for everything:

Victor Bahl   Glenn Brewer   Hwa Chang
Krysta Chauncey   Henri Dubois-Fèrriere
Prabal Dutta   Kipp Dye, MSPT   Shane Eisenman
Rodrigo Fonseca   Omprakash Gnawali
Prem Gopalan   Mauricio Gutierrez   Daniel Halperin
Mark Handley   Dr. Michael Kane   Brad Karp
C. Emre Koksal   Bhaskar Krishnamachari
Phil Levis   Dr. Rui Xiong Mai
Panayiotis Mavrommatis   Mark Nesky
Max Poletto   Ganesh Ramaswamy   Rodrigo Rodrigues
Stefan Savage   Jennifer Schmitt   Y. C. Tay
Patrick Thiran   Alice Tzou   Chieh-Yih Wan
Alec Woo

⋆ ⋆ ⋆

My parents Hazel and Nigel Jamieson and my sister Kirsty Jamieson have given me love and support, for which I am so grateful and without which I would not have been able to finish this dissertation. I thank my parents for providing a strong intellectual foundation upon which this dissertation is built, and for fighting hard to make my higher education possible. I thank my sister for being there for me and being willing to listen.

I thank Denise Leung for her love, warmth, and emotional support. I have benefited from her kind spirit, effusive personality, and sense of humor, without which I would not have survived during the difficult times. I count myself as lucky to have found someone as self-sacrificing and supportive as her. As I finish this dissertation and she finishes her education, I look forward to our life together.

PREVIOUSLY-PUBLISHED WORK

Chapter 2 contains some revised material from two previous publications: "Mitigating Congestion in Wireless Sensor Networks," [49] with B. Hull and H. Balakrishnan in the proceedings of the 2004 ACM SenSys Conference, and "Understanding the Real-World Performance of Carrier Sense" [59] with B. Hull, A. Miu, and H. Balakrishnan in the proceedings of the 2005 ACM SIGCOMM E-WIND Workshop.

Chapters 3, 4, and 5 significantly revise a previous publication: "PPR: Partial Packet Recovery for Wireless Networks" [58] with H. Balakrishnan in the proceedings of the 2007 ACM SIGCOMM Conference.

Chapter 6 contains a small amount of material from a previous publication: "Harnessing Exposed Terminals in Wireless Networks" [123] with M. Vutukuru and H. Balakrishnan in the proceedings of the 2008 USENIX NSDI Conference.

Contents

List of Figures 13

List of Tables 17

1 Introduction 19
1.1 An introduction to the problem 19
1.2 Approach 23
1.3 Contributions and results 24
1.4 Contents of this dissertation 26

2 Background 29
2.1 Multiple access in Ethernet and ALOHANET 30
2.2 The move to wireless networks 31
2.3 Other perspectives 34
2.4 A closer look at CSMA 36
2.5 An experimental evaluation of CSMA 37
2.6 Wireless sensornet "congestion collapse" 44
2.7 Related work 48

3 The SoftPHY Physical-Layer Interface 51
3.1 Design overview 52
3.2 Physical-layer design 54
3.3 Evaluation of SoftPHY hints 65

4 Postamble-based Synchronization 73
4.1 Synchronization in digital communications 73
4.2 Implementation 75
4.3 The need for a postamble 80
4.4 The postamble-based approach 81
4.5 Codeword-level synchronization 83
4.6 Related work 84

5 Reliable Retransmissions with SoftPHY **87**
5.1 The PP-ARQ protocol 88
5.2 Tuning feedback with dynamic programming 95
5.3 Asynchronous packet streaming 100
5.4 Implementation 105
5.5 Evaluation of PP-ARQ 107
5.6 Related work 130

6 Conclusion **135**
6.1 Other Uses of SoftPHY 135
6.2 Final thoughts 138

Bibliography **140**

List of Figures

1-1 A two-packet collision 20
1-2 Interference significantly raises bit error rate 21
1-3 The status quo "wired" abstraction 22
1-4 Bits in a packet do not share fate 23
1-5 Contributions of this dissertation 25

2-1 Two ways carrier sense can fail 32
2-2 Direct sequence spread spectrum modulation 35
2-3 Matched filter for detection in noise 37
2-4 Carrier sense improves links 40
2-5 Carrier sense misses opportunities for spatial reuse 41
2-6 Energy-detect carrier sense yields good links 42
2-7 How energy-detect carrier sense prevents reuse 43
2-8 Carrier sense slows down sensor nodes 44
2-9 Carrier sense slows down sensor nodes 45
2-10 Congestion collapse: channel and buffer loss 46
2-11 Congestion collapse: per-node delivery 47
2-12 Congestion collapse: bits received per unit energy 47

3-1 SoftPHY in an uncoded receiver 55
3-2 Convolutional encoder for IEEE 802.11a 59
3-3 Trellis diagram of an IEEE 802.11a-like code 60
3-4 Comparison between the Viterbi and BCJR algorithms . . . 61
3-5 From APPs to SoftPHY hints 63
3-6 BCJR-based SoftPHY confidence in 802.11a 64
3-7 Zigbee testbed map 65
3-8 SoftPHY hint error rates 67
3-9 Contiguous "miss" length distribution 68
3-10 BER v. SINR in a "quiet" network 69
3-11 SoftPHY hints at marginal SINR in a quiet network. 70

4-1 Transmitter-receiver timescale mapping 75

4-2 OFDM preamble search experiment 76
4-3 Mueller and Müller Zigbee synchronizer 77
4-4 Mehlan, Chan, Meyr (MCM) Zigbee synchronizer 78
4-5 MCM preamble search experiment 80
4-6 Overlapping packets motivate a need for a postamble 80
4-7 Packet layout for postamble-based synchronization 81
4-8 The postamble-based synchronization algorithm 82
4-9 SoftPHY recovers bits when synchronization is temporarily lost and then regained 84

5-1 PP-ARQ protocol overview 89
5-2 Run length representation of a packet 91
5-3 PP-ARQ feedback byte layout 93
5-4 PP-ARQ forward-link transmission byte layout 93
5-5 PP-ARQ forward-link retransmission byte layout 94
5-6 PP-ARQ forward-link data fragment byte layout 94
5-7 PP-ARQ forward-link bad fragment data byte layout 95
5-8 PP-ARQ dynamic programming example packet reception . 96
5-9 BOTTOM-UP-PP-ARQ-DP pseudocode 98
5-10 PP-ARQ dynamic programming example 98
5-11 GEN-FEEDBACK pseudocode 99
5-12 Streaming PP-ARQ forward-link packet format 100
5-13 Streaming PP-ARQ reverse-link packet format 101
5-14 PP-ARQ with streaming: feedback channel loss 101
5-15 PP-ARQ with streaming 102
5-16 PP-ARQ-SEND pseudocode 103
5-17 PP-ARQ-RECEIVE pseudocode 104
5-18 An excerpt from a Zigbee codeword-level packet trace . . . 106
5-19 Zigbee testbed map 109
5-20 Selected Zigbee and WiFi channels in the 2.4 GHz band . . 109
5-21 The per-fragment checksum approach 111
5-22 Experimental length of Zigbee error bursts 112
5-23 Experimental location of Zigbee errors 113
5-24 Aggregate PP-ARQ v. status quo throughput comparison . . 114
5-25 Per-link PP-ARQ v. status quo throughput comparison . . . 115
5-26 Feedback data overhead distribution 116
5-27 Retransmission size distribution 119
5-28 Isolating the impact of the postamble on throughput 120
5-29 Most packet receptions contain few "runs" 121
5-30 Throughput vs. fragment size 122

5-31 Per-link throughput distribution 123
5-32 Link-by-link throughput scatter plot 124
5-33 Per-link equivalent frame delivery rate distribution 125
5-34 Per-link equivalent frame delivery rate, carrier sense off . . . 126
5-35 Per-link equivalent frame delivery rate at high load 127
5-36 Aggregate trace-driven throughput 127
5-37 Retransmission size comparison 128
5-38 Aggregate throughput under varying offered loads 129
5-39 Impact of carrier sense on throughput 129
5-40 Coding and interleaving data 131
5-41 Incremental redundancy Hybrid ARQ 132

6-1 An example transmission from **S** to **R** with three abstract sender cases: an in-range but conflicting sender **CS**, an exposed sender **ES**, and a hidden sender **HS**. 136

List of Tables

2.1 Energy detect carrier sense results from a large scale Mica2 testbed 39

3.1 Experimental regimes for evaluating SoftPHY at marginal SINR. 70

5.1 Roadmap of experimental results 108

5.2 A summary of the overheads that PP-ARQ incurs 118

1

Introduction

OVER THE PAST CENTURY, the development of wireless technology has fundamentally changed the way we obtain information and communicate with others. Wireless technology profoundly impacts our personal, social, and professional lives: we now take for granted the presence of cellular telephones, wireless-enabled mobile computers, and wireless handheld devices.

In this century, all signs point to the continued existence and expanded influence of wireless technology in our daily lives. The wireless telecommunications and data industries are large and continue to experience rapid growth [100, 105] as demand for more and richer wireless content continues. Technological innovation is proceeding at a swift pace in both academia and industry, with the introduction of new technologies such as software-defined radio [25, 80, 99], mesh networks [2], and advances in high-speed, low-power radio frequency circuit designs [68, 92].

However, there is a factor pushing against the growth of wireless networks: wireless spectrum is fundamentally a shared and scarce resource. Proliferating numbers of nodes and demand from each node act to increase interference in the network, driving the performance each node experiences downwards. In this work we focus on techniques for increasing network throughput, allowing network designers to provide more and better service with fewer resources.

1.1 AN INTRODUCTION TO THE PROBLEM

Bit errors over wireless channels occur when the signal to interference and noise ratio (SINR) is not high enough to decode information correctly. In ad-

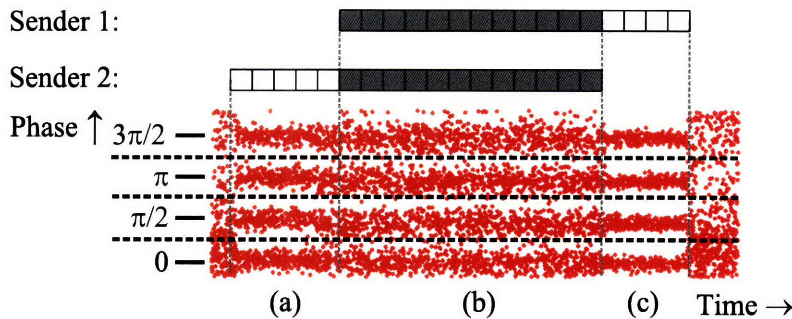


FIGURE 1-1—A collision between two packets, each sent from a different sender, as viewed by a nearby receiver. We plot the phase of each sample the receiver measures. The bits in ranges (a) and (c) are likely correct, while the bits in range (b) are more likely errored, as evidenced by the higher variance of the symbol samples shown below the packets.

dition to background noise, poor SINR arises from *interference* caused by one or more concurrent transmissions in the network, and varies in time even within a single packet transmission. The reason for this is that the wireless medium is fundamentally a shared medium, with some amount of energy from each transmission reaching not just its intended receiver, but other receivers in its vicinity. We can observe interference qualitatively in the packet "collision" of Figure 1-1 where the samples of Sender 1's signal in region (b) are much noisier than Sender 1's samples in region (c), causing many of the former samples to be "pushed" above or below the dashed horizontal decision boundaries shown in the figure, resulting in bit errors.

Modern wireless systems use a variety of sophisticated physical-layer techniques to combat channel impairments such as background noise, channel attenuation, and multipath fading. Examples of these techniques include advanced modulation techniques such as Orthogonal Frequency Domain Modulation (OFDM) and spread-spectrum modulation [6,41,97], and error-control coding [18, 70]. These techniques make good individual links between pairs of nodes possible, but they fall short of our goal of mitigating the effects of interference in a network viewed as a whole.

To see why quantitatively, consider the bit error rate versus SINR curve shown in Figure 1-2.[1] The physical layer techniques mentioned earlier (including channel coding in this example) can drive link bit error rate down

[1]Data source: Harris Corp. 802.11b baseband processor at 5 Mbits/s [45].

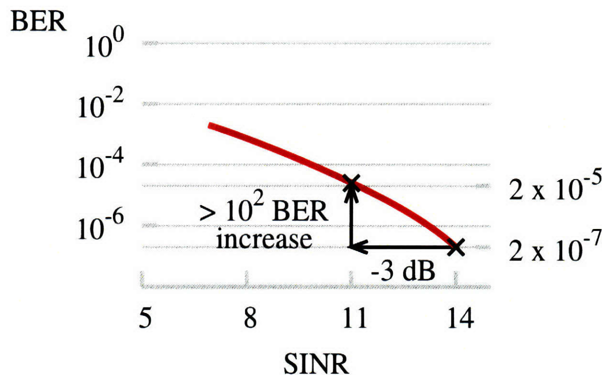


FIGURE 1-2—A 3 dB loss in signal to interference plus noise ratio (SINR) from an interfering transmission of about the same strength as the intended transmission results in a two-order increase in bit error rate.

to 10^{-6} or below, as indicated by the rightmost data point on the curve of Figure 1-2.

However, consider what happens when an interferer begins sending in the vicinity and its transmission arrives at the receiver with approximately the same received signal strength as the transmission the receiver intends to receive. The interfering transmission reduces the SINR of the intended transmission by approximately 3 dB,[2] resulting in the approximate two-order increase in bit error rate at the left-most data point in Figure 1-2. The resulting bit error rate, 2×10^{-5}, yields (on average) one bit error in each 1500 byte packet, even after the channel coding that 802.11b applies. This reflects a catastrophic operating point for a network, because current wireless protocols discard packets containing any bit error, causing network throughput to drop.

★ ★ ★

Of course, it is a well-known fact that the wireless medium is shared, deriving from the principle of superposition of electromagnetic waves from the electromagnetic theory [98]. As a result, many ways of sharing the wireless medium have been invented. These techniques also sometimes go by

[2]The 3 dB reduction in SINR holds in a regime where the power of the interfering transmission is significantly larger than the power of the background noise in the network.

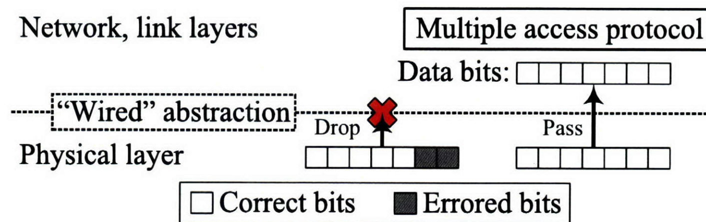


FIGURE 1-3—Status quo physical layers implement a "wired" abstraction which drops frames with any number of bit errors, and passes only the data bits themselves up to higher layers.

the names "medium access control" or "multiple access" protocols. One of the most popular techniques for sharing the wireless medium is called carrier sense multiple access (CSMA), and we describe it in detail in the next chapter.

As designers of large, busy wireless networks, it helps to examine the problem at a larger scope. To maximize the capacity of a multihop wireless mesh network, we desire to both increase the amount of spatial reuse in the network and decrease the amount of interference that each transmission experiences. We also note a tension between these two goals: increasing spatial reuse means permitting more concurrency between transmissions, while decreasing concurrency reduces interference between transmissions. Medium access control protocols, as we will see in the next chapter, often make mistakes when balancing these two factors, and the result is either errored packets or a lack of concurrency in the network.

The wired abstraction. The way that most current wireless systems [50, 51, 52, 53, 54] handle these inevitable mistakes is to interpose what we will refer to as a *wired abstraction* over the physical layer, represented by the dotted line in Figure 1-3. The wired abstraction's purpose is to make the error-prone wireless physical layer appear error-free to higher layers. The mechanism it uses to accomplish this is to apply some amount of channel coding to the link, but then to simply drop the decoded frame if any part of it is errored.

When the wired abstraction drops a packet containing one or more bit errors, higher layers generally need to retransmit that packet. Retransmitting entire packets works well over, for example, wired networks where bit-level corruption is rare and a packet loss implies that all the bits of the packet

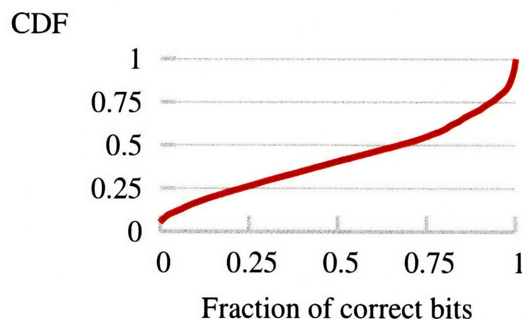


FIGURE 1-4—Bits in a packet do not share fate: the distribution of correct bits in errored packets is roughly uniform, indicating that many bits in packets whose checksum fails are in fact correct.

were lost (e.g., due to a queue overflow in a switch). In these situations, we say that the bits in a packet "share fate" with each other. In wireless networks, however, the bits in a packet most often do not share fate. Because wireless channels are hard to model and predict, instead often only a small number of bits in a packet are in error while the rest are correct. Data shown in Figure 1-4 from a large spread-spectrum network with carrier sense multiple access enabled underscores this point. Consider the fraction of bits in a packet that are correct when at least one bit is errored. We find from the figure that this quantity is uniformly distributed across packets. The wired abstraction discards those correct bits, wasting network capacity.

To summarize the problem: even with the use of error-control coding and robust modulations, current systems are built on sharing protocols that make mistakes in the face of interfering transmissions, and rely on link-layer retransmissions to present the wired abstraction to higher layers. Thus bit errors cause wasteful retransmissions of the entire packet when the wired abstraction drops it. As a result, presenting the wired abstraction to higher layers generally entails sacrificing significant capacity.

1.2 APPROACH

This dissertation presents the design, implementation, and evaluation of techniques to improve aggregate network throughput by reducing the number of bits transmitted.

Key insight. The key idea that impacts each of the architectural components and protocols we propose in this dissertation is to shift the way we reason about wireless networks, at all layers of the network stack, from a packet-centered view to one that reasons about individual *symbols*. In the context of our discussion, we use the term "symbol" to mean the unit of information granularity at which the physical layer makes decisions. This concept may correspond to actual physical layer symbols e.g. in the case of an uncoded transmission, or it may correspond to codewords in a coded transmission. Thus instead of presenting the wired abstraction to higher layers, we propose a physical layer design that occasionally lets errors occur and delivers *partial packets* up to higher layers.

There are several challenges in realizing this vision and making it practically useful.

1. How can a receiver tell which bits are correct and which are not?

2. Since most physical layers require the receiver to synchronize with the sender on a preamble before decoding a packet's contents, wouldn't any corruption to the preamble (caused, for instance, by a packet collision from another transmission) greatly diminish the potential benefits of the proposed scheme?

3. How exactly can we use the bits we recover from partially-incorrect packets to improve end-to-end performance of various network- and link-layer protocols?

1.3 CONTRIBUTIONS AND RESULTS

Our design contributions, shown in Figure 1-5, fall into two broad categories: architectural design and protocol design. Our key architectural contribution is a new interface for the physical layer called the *SoftPHY interface* and its implementation on a few common radio standards.

1.3.1 Design contributions

The SoftPHY interface. The SoftPHY interface allows the receiver to determine, with no additional feedback or information from the sender, which bits are likely to be correct in any given packet reception using hints from the physical layer. The key insight in SoftPHY is that the physical layer should pass up information about how close each received symbol or codeword was to the symbol or codeword the physical layer decided upon. Higher layers

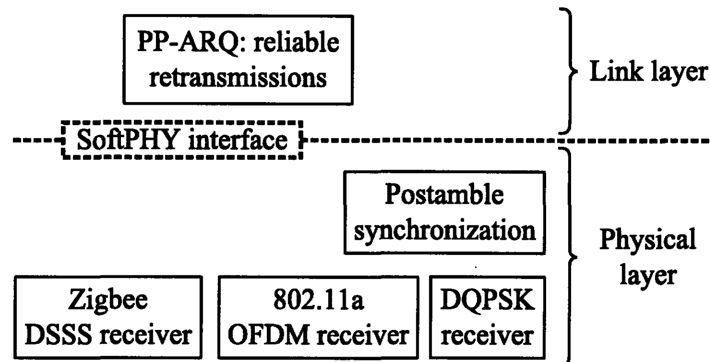


FIGURE 1-5—Contributions of this dissertation. At the physical layer, we have implemented three distinct receivers for 802.11a, Zigbee, and DQPSK, along with the postamble decoding techniques described in Chapter 4. We have implemented the SoftPHY interface of Chapter 3 for each receiver. At the link layer, we have designed and developed PP-ARQ, a protocol for reliable retransmissions using the SoftPHY interface.

can then use this information as a hint, *independent* of the underlying details in the physical layer. We show that SoftPHY has applications across many higher layers of the networking stack.

Postamble decoding. Postamble decoding allows a receiver to receive and decode bits correctly even from packets whose preambles are undetectable due to interference from other transmissions or noise. The key idea is to replicate the information in the preamble and packet header in a postamble and a packet trailer, allowing a receiver to synchronize on the postamble and then "roll back" in time to recover data that was previously impossible to decode.

The PP-ARQ protocol. Using the above two techniques, we have designed *partial packet ARQ* (*PP-ARQ*), a link-layer reliable retransmission protocol in which the receiver compactly requests the retransmission of only the select portions of a packet where there are bits likely to be wrong. In response, the sender retransmits the bits and checksums for those ranges, so that the receiver can eventually be certain that all the bits in the packet are correct. The receiver's request encoding uses a dynamic programming algorithm that minimizes the expected bit overhead of communicating this feedback, balancing that against the cost of the sender retransmitting bits already received correctly. PP-ARQ uses windowed asynchronous stream-

ing techniques to recover from feedback loss and amortize packet header overhead.

1.3.2 Experimental contributions

Implementation work. We have implemented and integrated each of our three contributions for IEEE 802.15.4 [54], the Zigbee standard, and our implementation is compatible with that specification (see Section 3.2.2, page 57). The SoftPHY and postamble decoding architecture contributions can recover partial packets from unmodified Zigbee senders, while the PP-ARQ protocol requires sender side modifications.

We have also implemented the SoftPHY interface for a convolutionally-coded OFDM system with the same structure as IEEE 802.11a [52] (see Section 3.2.3, page 63), and in an uncoded DQPSK receiver (Section 3.2.1, page 54).

Performance results. Our techniques can improve performance in both access point-based networks and wireless mesh networks. Chapter 5 shows experimental results that confirm this: in that chapter, we describe a 31-node indoor testbed consisting of Telos motes with 2.4 GHz Zigbee radios from Chipcon and six GNU Radio nodes. Our results show approximate 2× gains over the status quo in aggregate end-to-end throughput using PP-ARQ.

We compare our techniques to other ways of determining which bits are likely to be correct, such as fragmented packet checksums. Finally, we compare PP-ARQ to an idealized protocol that has no overhead and unlimited knowledge about which bits were received correctly. We analyze and fully account for the gap between PP-ARQ's performance and this idealized protocol's performance.

The underlying premise in this work is that significant performance gains can be obtained by the combination of a more aggressive, higher-rate physical layer and being more flexible about the granularity of error recovery in wireless networks. In Chapter 6, we give reasons why SoftPHY may enable even bigger performance gains in future work.

1.4 CONTENTS OF THIS DISSERTATION

In the next chapter we present background information on the mechanism of carrier sense in the same type of wireless networks we evaluate our contributions. In Chapter 3 we propose a new interface for the physical layer

called the SoftPHY interface. Then in Chapter 4 we discuss the packet synchronization and detection problem, and introduce postamble decoding, a new method for more robust packet detection and synchronization.

The following two chapters, 5 and 6, describe novel uses of the SoftPHY interface. Chapter 5 describes a novel protocol called PP-ARQ for retransmitting only the bits in a packet most likely in error. In the following chapter, we describe other uses of the SoftPHY interface in medium access control, opportunistic bit-wise forwarding in a mesh network, and bit-rate adaptation. Chapter 6 concludes the dissertation, examining alternate approaches and reflecting on the choices we made in the design of the system.

2

Background

IN CHAPTER 1 WE NOTED that the wired abstraction constrains performance in wireless networks. This is due in large part to an unfortunate interaction between the wired abstraction and protocols to share the wireless medium between different senders. Situated at the link layer, the job of the medium access control (MAC) protocol is to share the wireless medium between each sender. MAC mistakes and tradeoffs, however, result in either errored bits in packets or a loss of concurrency in the network. Both types of mistakes lead to a loss of network capacity.

Chapter overview. In this chapter, we survey protocols to share the wireless medium, tracing the development of the carrier sense multiple access used in many of today's wireless and mesh networks. We then briefly discuss other ways of sharing the wireless medium before taking a quantitative look at the mistakes that carrier sense makes, using two different testbed configurations. The first is a 60-node wireless sensor network communicating with Chipcon CC1000 [116] narrowband FM radios. These radios have a data rate of 38.4 Kbits/s. The second is a small-scale 802.11 testbed using Atheros 5212 OFDM radios. Our experimental data show that while carrier sense improves link qualities at all traffic loads, it leaves room for the performance improvements we leverage in later chapters. We conclude this chapter with a discussion of the problems that wireless sensornet traffic patterns in particular create.

2.1 MULTIPLE ACCESS IN ETHERNET AND ALOHANET

The ALOHANET packet radio network was designed by Norman Abramson at the University of Hawaii in the late 1960's. Abramson saw the need to interconnect the seven geographically-disjoint campuses of the University of Hawaii together. ALOHANET served as the means of sharing the computer systems at the main campus with the six other campuses of the university, all within a radius of about 300 km of each other. The intent was for computer users to use ALOHANET for remote teletype access to computer systems located at the main campus.

The sharing protocol that ALOHANET ran, ALOHA [1], is one of the first examples of a communications network that uses a shared communications medium (in this case, the wireless channel) for communication. In ALOHA, just transmit a packet when one is ready to send. Checksums attached to each packet implement the wired abstraction concept we introduced in Chapter 1, so stations discard corrupted packets. There are well-known throughput analyses of ALOHA's performance [9, 32, 66] under a number of theoretical assumptions. ALOHA has inspired many subsequent developments, both practical and theoretical, which we now examine.

One key development ALOHANET helped to inspire [115] is the wired Ethernet [76] local-area computer network. The original Ethernet used one stretch of coaxial cable, into which physical taps were placed, one tap for each computer connecting to the Ethernet. Ethernet transceivers listen to the shared medium before every transmission to determine if there is a packet transmission ongoing. If there is a packet transmission ongoing, then the transceiver *defers* to the ongoing transmission; otherwise it begins its own. This mechanism is called *carrier sense*, and we discuss it in more detail below.

Collision avoidance and detection in Ethernet. Ethernet is a non-persistent CSMA protocol [9]. In such protocols, the time immediately after each transmission is divided into CW *contention slots*, whose duration is several orders of magnitude smaller than the time it takes to send a data packet. Immediately after a transmission or collision, each station picks a random contention slot $r \in [1, CW]$. During the contention slots prior to r, each station carrier-senses the medium, deferring its pending transmission if it hears the beginning of another transmission. At contention slot r, the station begins its transmission. If two nodes pick the same slot, they both transmit at the same time, causing a collision. When this happens, the colliding nodes double their value of CW. This is known as *binary exponential*

backoff (BEB). By increasing CW, BEB protocols attempt to adapt to the current active population size to make a collision-free transmission more likely.

Ethernet also has a *collision detection* mechanism that can detect the presence of concurrent transmissions while one is ongoing. The transceiver listens to the reflection of its transmission off the end of the coaxial Ethernet wire. It then compares the reflection to an appropriately-delayed version of the transmitted signal and if the two differ, raises the "collision" line [77]. Saltzer [104] has observed that since each transceiver's rate of current injection onto the wire is a known constant, the "collision detector" could also examine DC voltage on the wire, which would increase in the presence of more than one concurrent transmission. In a wireless transceiver, a similar collision detector is much harder to design because antenna loss and signal attenuation in space makes the received signal tens of decibels lower than the transmitted signal in power.

Enforcing consensus in Ethernet. When an Ethernet station detects a collision, it jams with wire with a carrier signal for one round-trip time, to ensure that all other stations on the wire agree that there was a collision. Hence, the Ethernet makes no attempt to reuse different spatial extents of the wire medium. In contrast, and again because of attenuation, wireless protocols have an opportunity[1] [109] for *spatial reuse*: using different parts of space for simultaneous communications. We will see that this complicates the problem of achieving good wireless network utilization.

2.2 THE MOVE TO WIRELESS NETWORKS

We now review how, with some degree of success, mechanisms from the wireless ALOHANET and wired Ethernet were translated to and evolved in wireless networks. We begin with a key observation made by Karn [63] and affirmed by Bharghavan et al. [10].

2.2.1 Carrier sense is a heuristic in wireless

In a wireless network, unlike in the Ethernet, the location of transmitters and receivers matters, because the probability of receiving a packet correctly depends on the SINR at the receiver. Thus, the transmitter's carrier sense line

[1]Though some wireless protocols, such as FAMA-PJ [29] use the same jamming mechanism to eliminate collisions, at the expense of some spatial reuse.

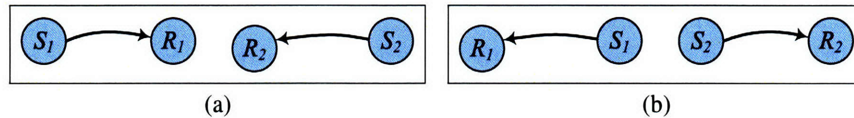


FIGURE 2-1—(a) a "hidden terminal" node configuration: carrier sense may not schedule the two indicated transmissions to defer to each another, causing bit errors. (b) an "exposed terminal" node configuration: carrier sense may schedule the two indicated transmissions to defer to each other, whereas they could transmit simultaneously with error, doubling throughput.

is at best a guess about conditions at the receiver. This guess can be correct if the receiver and sender are close enough that they experience similar noise and interference conditions. However in many cases, no correlation exists between channel conditions at the sender and at the receiver. This lack of correlation is often due to exposed and hidden terminals, the aggregate effect of distant nodes raising the noise floor, and capture.

Hidden terminals. First, carrier sense can permit transmissions to corrupt each other when the senders are out of carrier sensing range of each other, as in Figure 2-1(a). This is sometimes called a *hidden terminal* situation.

Exposed terminals. Second, it can prevent senders (e.g., S_1 and S_2 in Figure 2-1(b)) from transmitting simultaneously when their intended destinations have a lower levels of mutual interference—an *exposed terminal* situation. Unfortunately, carrier sense would only allow one transmission to take place: whichever node lost the CSMA contention period would sense a busy channel and wait for the other node's transmission to complete.

Capture. In addition, receivers can sometimes decode transmissions even in the presence of relatively strong interfering transmissions within the same frequency band [112, 127]. In other words, this means that concurrent transmissions are possible by a set of nodes well within each others' transmission range. This differs from the carrier sense assumption that only one node should be transmitting in the receiver's radio neighborhood. This suggests that simply extending the carrier sense mechanism to the receiver does not solve the problem.

Distant transmissions. Finally, carrier sense may be a poor predictor of transmission success if interference comes from a large number of distant

nodes rather than a few local neighbors [109]. When interference is local and nodes are within each other's transmission range, carrier sense or an RTS/CTS exchange may be a good method of contending for the channel. However, because a node's interference range is much larger than its transmission range, distant transmitters can easily impact local transmissions. In aggregate, these distant transmitters raise the overall "noise floor" of the network, reducing link quality. Carrier sense as described in Section 2.4 cannot mitigate this type of interference.

2.2.2 The evolution of multiple-access in wireless networks

Noting the above problems with carrier sense, Karn proposed discarding the carrier sense signal from radio transceivers in the MACA [63] protocol. MACA instead relies on a reservation strategy called the *ready-to-send/clear-to-send exchange* (RTS/CTS exchange) to protect against interfering transmissions.[2] When a station has a packet to send, it first sends a short RTS message indicating its intention to send data. If and when the intended receiver decodes the RTS, it sends a CTS message indicating its intention to receive. Both the RTS and the CTS messages contain the duration of the data transmission. If any other station not involved in the transmission decodes either an RTS or a CTS message, it defers its transmissions for the duration of the ongoing data transmission. For the RTS/CTS exchange to be worth the overhead it incurs, the data messages have to be several orders of magnitude longer than the control[3] packets.

MACAW [10] revised MACA to include Ethernet-like non-persistent collision avoidance, combined with a mechanism for nodes to learn about contention levels from other nodes in the wireless network. Instead of BEB, MACAW uses a multiplicative-increase, linear decrease (MILD) algorithm to adjust the contention window, and copies contention window values from station to station.

Subsequently to MACAW, and in contradiction with what MACA and MACAW suggest, Fullmer and Garcia-Luna-Aceves proposed the FAMA family of MAC protocols, which combine Ethernet's non-persistent CSMA with the RTS/CTS exchange to yield better performance in the presence of hidden terminals [30, 31].

---

[2]This reservation strategy predates MACA itself; one of the first protocols to propose the RTS/CTS exchange was split-channel reservation multiple access (SRMA) [119]. Apple Computer Inc.'s Appletalk also used a similar reservation strategy in the wired domain.

[3]RTS, CTS, or in subsequent protocol descriptions, acknowledgment packets.

Wireless local-area networks. The development of commodity wireless local-area networking chipsets mirrors the developments in the research literature. The IEEE 802.11 standard for wireless LANs [50], parts of the the ETSI HIPERLAN [26] MAC protocol, and various sensornet MAC layers [94, 133] use carrier sense with non-persistent collision avoidance, and most use BEB to adjust their contention levels to the amount of actual contention in the network.

One drawback of RTS/CTS is that makes the tacit assumption of "all-or-nothing" interference where either a transmission interferes with the intended one and the packet is useless, or the transmission is completely independent of the other. Keeping in mind the SINR model of interference above, however, we see that the all-or-nothing model poorly reflects reality. Furthermore, there are situations where even under the assumption of all-or-nothing interference, the RTS/CTS exchange does not suffice to guarantee collision-free transmissions [31]. Finally, in a busy building-wide (network diameter much larger than a radio range) 802.11 infrastructure network, Judd finds that hidden terminals are much less common than exposed terminals [61]. Furthermore, two randomly-chosen clients are as likely to be exposed terminals with respect to each other as they are to connect to the same access point. This and the bandwidth overhead associated with control messages could be why RTS/CTS is almost never used in extant 802.11 networks.

At their core, these randomized carrier sense multiple access (CSMA)-based MAC protocols attempt to adapt to the active population size of contending nodes. Typically, each node maintains a slotted *contention window* with collisions (i.e., unsuccessful transmissions) causing the window to grow in size, and successful transmissions causing it to shrink. Each node transmits data at a slot picked uniformly at random within the current contention window.

2.3 OTHER PERSPECTIVES

Of course, CSMA is not the only way of sharing the wireless medium. Senders may instead use *time-division multiplexing* (TDM), dividing time into *timeslots* one data packet's duration in length (in general, TDM assumes equally-sized data packets). Each sender then gets a transmission opportunity in timeslots allocated to it by some mechanism, and must remain silent in the remainder of timeslots. In general, TDM is highly suboptimal for bursty traffic workloads [118], but performs rather well for constant-rate workloads where there is a constant demand from each sender. There are

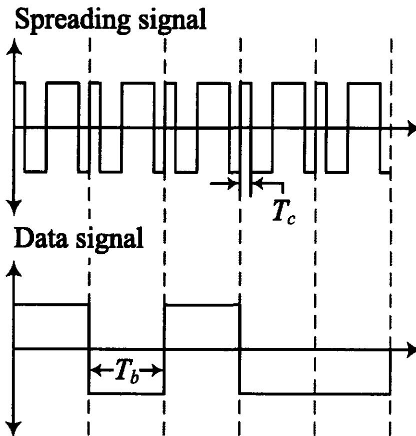


FIGURE 2-2—(Proakis [97]) Direct sequence spread spectrum (DSSS) modulation of a data signal with bit time T_b. DSSS spreads the data signal in frequency by multiplying the data signal (lower) with a spreading *chip sequence* (upper) of period T_b and chip time T_c.

proposals that mix TDM and CSMA medium access [82, 101], and proposals for allocating TDM time-slots that try to maximize throughput [16].

Senders may also choose different frequency bands on which to transmit; this is called *frequency-division multiplexing* (FDM). Both TDM and FDM have the drawback of leaving spectrum unused when traffic patterns are bursty.

As an alternative to simple division of time or frequency senders may simultaneously transmit waveforms in the same frequency bands, using signals that can be decoded independently from each other under some conditions. One way of accomplishing this is called *code division multiple access* (CDMA). CDMA uses *direct sequence spread spectrum* modulation (Figure 2-2) where senders use mutually-orthogonal spreading sequences. This idea, along with power control between each mobile and the base station and a carefully-planned base station deployment allow CDMA cellular phone networks to achieve efficient sharing of the wireless medium. In fact, from an information-theoretic perspective, CDMA with multi-user decoding (Section 2.7) comes closest to achieving wireless capacity in the two-user case [32, 120]. Unfortunately, mesh networks in general lack the element of careful planning that goes into cellular phone networks, making it difficult to adapt CDMA for this purpose. Most mesh networks instead use CSMA because of its simplicity.

2.4 A CLOSER LOOK AT CSMA

In this section we examine the theory and mechanisms behind the carrier sense used in Zigbee and 802.11 radios. In the course of this development, we give some insight into why carrier sense sometimes makes channel access decisions that result in lower aggregate throughput in the network.

There are two primary ways that modern radios implement carrier sense; first, by computing a correlation between the input signal and the preamble, and second, by measuring changes in received energy.

2.4.1 Signal detection theory

This technique relies on the inclusion of known data in each packet; this data is referred to as *preamble*. In Chapter 4 we provide more detail on the synchronization process (which is needed for the correlation computation) and additional details specific to Zigbee and 802.11. We now briefly survey the theory behind this method, adapting the discussion in Oppenheim and Verghese [87] to the specifics of our problem.

Suppose we observe an input signal $y[k]$ from the RF downconverter. We want to decide between two hypotheses:

$$\begin{aligned} H_0 \text{ (preamble absent) } &: \; y[k] = w[k] \\ H_1 \text{ (preamble present) } &: \; y[k] = p[k] + w[k] \end{aligned} \tag{2.1}$$

Under hypothesis H_0, the input signal is simply noise $w[k]$ with no preamble transmission present, and the carrier sense circuitry indicates "carrier sense free." Under hypothesis H_1, the input signal is the sum of noise and a preamble signal $p[k]$ of length L, and the carrier sense circuitry indicates "carrier sense busy."

Under certain assumptions on the noise statistics, it can be shown [6, 87, 121] that the decision rule to minimize the probability of error consists of the following steps, as shown in Figure 2-3. First, the input signal $y[k]$ passes through a filter with response equal to a time-reversed version of the preamble. Then, a sampler measures the output of the filter just after the preamble, at time $t = L$. Finally, a threshold device compares the sampled output against a threshold γ.

There are two key points to note from the foregoing discussion. First, the filtering step simply computes a correlation between the preamble and the input signal, since the output of the filter $\rho[k] = \sum_i y[i]h[k-i] = \sum_i y[k]p[k]$. We make use of these concepts in our discussion of postamble decoding in Chapter 4.

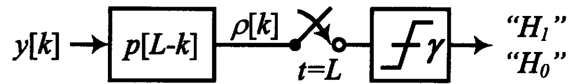


FIGURE 2-3—Sampling and thresholding the output of a linear, time-invariant filter with impulse response matched to the preamble yields minimum probability of error detection under certain assumptions on the noise statistics.

Second, there are two types of errors that our decision process can make: declaring "H_0" when in fact a preamble is present (a *missed preamble*), and declaring "H_1" (a *false alarm*) when in fact no preamble is present. The decision process compares the sampled correlation to a threshold γ, which can be adjusted to trade off between the two types of error. Raising γ increases the probability of a miss, which increases the number of hidden terminals carrier sense fails to detect (see Section 2.2.1). Lowering γ increases the probability of a false alarm, which increases the number of exposed terminal pairs carrier sense prohibits from transmitting at the same time (again see Section 2.2.1).

2.4.2 Energy detection

The *Energy detect* indicator is a carrier sense mechanism common to many extant radios. It is based on signal strength readings obtained from the radio front end. Over the periods that there are no incoming transmissions, senders time-average instantaneous signal strength readings into a quantity called *squelch* (σ). Squelch can be interpreted the "noise floor" of the network: the signal strength of background noise. Just before a transmission, the sender makes its carrier sense decision with a comparison between ρ and σ. If $\rho > \sigma$, then carrier sense is *busy*. Otherwise, carrier sense is *idle* and the sender may begin transmission. Alternately, some radios use a fixed threshold for σ, commonly referred to as the *carrier sense threshold* [114].

2.5 AN EXPERIMENTAL EVALUATION OF CSMA

As discussed above, carrier sense is a fundamental part of most wireless local area- and sensor network radios. As increasing numbers of users and more demanding applications push wireless networks to their capacity limits, the efficacy of the carrier sense mechanism becomes a key factor in determining wireless network capacity.

In this section, we present experimental results from both a large, dense 60-node sensor network deployment and a small-scale 802.11 deployment. Our results quantify how well carrier sense works and expose its limitations.

2.5.1 Experimental design

Sensornet design and implementation. Our first experimental setup is a 60-node indoor wireless sensor network testbed. Each node is a Crossbow Mica2, which has an Atmel ATmega128L microcontroller with 4 KB of RAM, 128 KB of flash, and a CC1000 radio [116]. The radio operates at 433 MHz, transmits at 38.4 Kbits/s, and uses frequency shift keying (FSK). Each node is attached to a Crossbow MIB600 interface board that provides both power and an Ethernet back channel for programming and data collection. We have deployed these nodes over an area of 16,076 square feet on one floor of our office building, with liberal coverage throughout the floor and a higher than average density in one corner of the floor. We use Motelab [126] to manage the testbed.

Our sensornet nodes run a variant of B-MAC [94] on the Chipcon CC-1000 [116] radio. B-MAC uses energy detection (see Section 2.4.2) to sense carrier. Each sender in the following experiments uses energy detect mechanism to sense carrier before transmitting, but in the runs without carrier sense, we modify B-MAC to record and ignore the carrier sense reading just before transmitting. In the runs with carrier sense, B-MAC records and follows the carrier sense reading, deferring transmission if carrier sense indicates busy.

802.11 experimental design. The second experimental setup is a small testbed consisting of three indoor 802.11 nodes in close proximity. Two nodes act as senders and are placed at about six meters apart. The receiver is placed approximately 12 meters from each of the senders. Each node is equipped with an Atheros 802.11 a/b/g combo card driven by the *madwifi* driver [72]. The radios use a punctured convolutional code of rate $R = \frac{1}{2}$ over OFDM modulation [53]. In contrast to the sensornet radios, the 802.11 radios use the signal correlation method (see Section 2.4.1) to detect carrier.

2.5.2 Carrier sense improves link delivery rates

Experimental setup. The purpose of this experiment is to evaluate the efficacy of carrier sense in a large, dense sensor network. In this experiment, the sensornet nodes transmit data in three traffic patterns:

Traffic pattern	Frequency of occurrence		Link delivery rate	
	CS free	CS busy	CS free	CS busy
One-by-one	56%	43%	81%	80%
All send 1 pkt/s	53	46	67	57
All send 4 pkts/s	22	78	46	14

TABLE 2.1—Frequency of carrier sense free and busy, and link delivery rate in a large-scale Mica2 testbed with the energy detection carrier sense method.

1. **One-by-one**: nodes take turns transmitting, each node transmitting only when no others are transmitting.

2. **1 pkt/s**: all nodes transmit data at a constant rate of one packet/sec per node, with a small random backoff before transmitting.

3. **4 pkts/s**: same as above, at the rate of 4 packets/sec per node.

Experimental results. We begin with a high-level view of our experimental results, measuring the fraction of time that carrier sense is busy or free under each of the above traffic workloads; table 2.1 summarizes the data. Reading the table, when nodes take turns transmitting one at a time, carrier sense reports that the medium is busy 43% of the time. Furthermore, in the aggregate, the one-by-one link delivery rate does not depend on what the energy detect carrier sense line indicates. This means that in a quiet network, carrier sense is generating false alarms (Section 2.4.1) at a rate of 43% in situations where the sender could transmit but will instead defer. At moderate traffic loads (one packet/s), we see the same effect, at a somewhat less-pronounced intensity. At moderate and high loads, link delivery rates plummet, because in this experiment, nodes are not deferring when carrier sense indicates busy.

Figure 2-4 shows the distribution of link delivery rates across links that are most likely to be useful to higher-level protocols (greater than 60% link delivery rate). The same trends shown in the figure hold for the bottom 60% links. Figure 2-4 shows that the probability of a single transmission succeeding increases when sensors perform carrier sense. As the amount of traffic in the network increases, however, link quality decreases sharply, even with carrier sense enabled. Furthermore, the more traffic in the network, the more carrier sense improves link quality. Carrier sense improves link quality by a small amount when each node offers 1 pps, but by a significant amount when each node offers 4 pps. At low loads, the channel is idle most of the

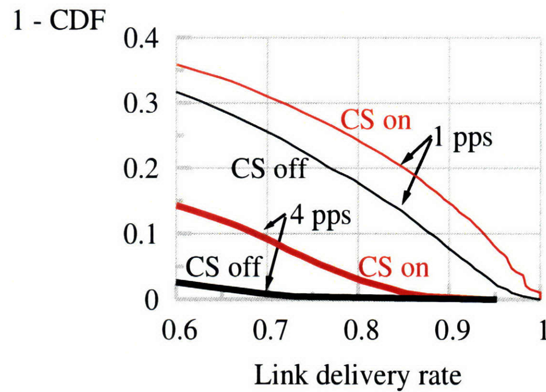


FIGURE 2-4—The complementary cumulative fraction of node pairs that achieve link delivery rates greater than 0.6 in the sensor network testbed. "CS on (off)" means that sensors perform (do not perform) carrier sense as described in Section 2.4.

time, and we hypothesize that therefore most losses are path losses, which carrier sense cannot prevent. At high loads, the channel is busy most of the time, and therefore carrier sense sometimes helps avoid the many collisions that occur.

2.5.3 Limitations of carrier sense

We now establish, in our 802.11 testbed, at which bit rates carrier sense is ineffective due to the capture effect described above in Section 2.2.1. We discover that carrier sense can be ineffective at low data rates when the capture effect is most prevalent. Consequently, the standard carrier sense algorithm can lead to many erroneous predictions and wasted transmission opportunities in practice.

We place two 802.11 senders (*A* and *B*) in close proximity such that they are well within communication range of each other at all bit rates. Because the typical carrier sense range is greater than communication range, the two senders are also well within carrier sensing range of each other. We modified the driver to disable randomized backoff, which gives the effect of disabling carrier sense and allows two senders to transmit packets simultaneously.

The results in Figure 2-5 show low packet delivery rates at high bit rates. Due to mutual interference from the simultaneous transmissions, the receiver fails to decode most data frames transmitted by either sender. As bit rate decreases, however, we observe that the receiver often captures one

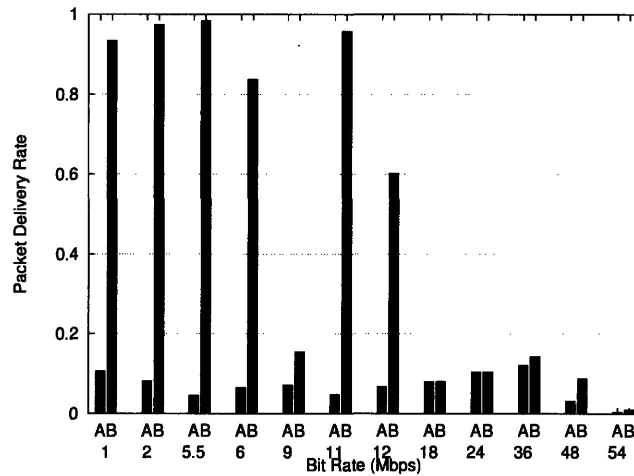


FIGURE 2-5—Packet delivery rate in the 802.11 network for two simultaneous senders, each transmitting at the saturation rate at the selected bit rate. Carrier sense is effectively disabled in this experiment.

of the senders (*B*), thus achieving a very high frame delivery rate from *B*. The degree of capture is surprisingly large at low bit rates: the link between sender *B* and the receiver achieves a delivery rate of over 80% for bit rates 1, 2, 5.5, and 6 Mbits/s.

The capture effect is attributed to the relative difference of the received signal strength between the two senders' transmitted frames. In our experiments, the average RSSIs of *A* and *B* are −57.2 dBm and −48.8 dBm respectively. In general, the probability of capture increases as the ratio between the RSSI of the captured signal and the RSSI of the drowned-out signal increases. Also, the minimum required signal ratio for capture to take place decreases as bit rate decreases.

Current carrier sense schemes are oblivious to the capture effect. Consequently, they mispredict transmission failures and wastes potential transmission opportunities when they exist. In our experiments, if randomized backoff were enabled (thereby, allowing carrier sense to take full effect) and we assume that the intended destination of *A* and *B*'s transmissions are different, then sender *B* would have deferred its transmissions due to sender *A*'s transmissions, even though the receiver could capture *B*'s transmissions at low bit rates.

We can improve network efficiency by designing a carrier sense mechanism such that it is capture-aware: it should make transmission deferral

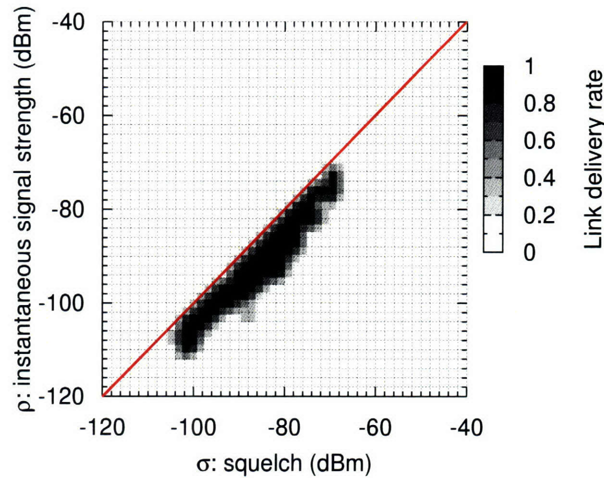


FIGURE 2-6—Packet delivery rate as a function of squelch and instantaneous signal strength just prior to transmission. All nodes send data at a rate of 4 pps/node with energy-detect carrier sense enabled.

decisions based on the bit rates being used and the packet delivery rates observed at all of the nearby receivers. For example, if A's intended receiver can capture B's transmissions, carrier sense should be used to defer B's transmission to prevent it from interfering with A's transmission. On the other hand, if A's intended receiver can tolerate a parallel transmission from B without significantly affecting A's delivery rate, carrier sense should be suppressed to make efficient use of the available transmission opportunities. In related work [123], we explore these ideas, showing a substantial performance gain.

How often do carrier-sensing senders miss transmission opportunities?

Recall that just before sending a packet, transmitters using the energy detect method of carrier sense (Section 2.4) compare their current squelch σ with the instantaneous signal strength ρ. To gain more insight into how and why energy detect works, we now examine delivery rates explicitly parameterized as functions of ρ and σ.

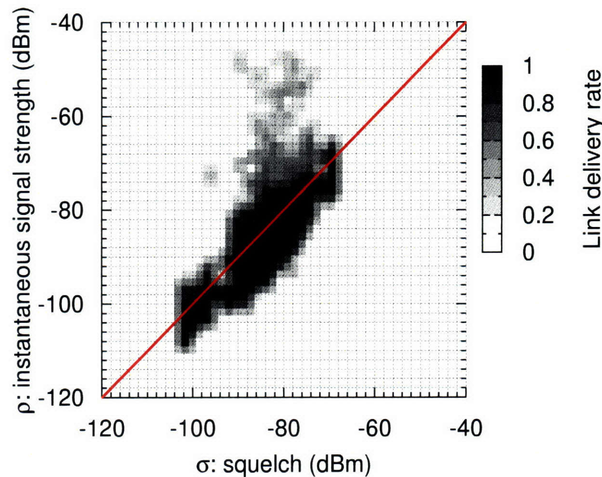


FIGURE 2-7—Packet delivery rate as a function of squelch and instantaneous signal strength just prior to transmission. All nodes send data at a rate of 4 pps/node, ignoring the energy-detect carrier sense decision. The dark points above the diagonal line are transmission opportunities that carrier sense misses.

In Figure 2-6, senders wait until $\rho < \sigma$ before transmitting. In Figure 2-7, senders record ρ and σ, but do not wait for ρ to fall below σ before transmitting. The figures show average link delivery rates as a function of ρ and σ. In both datasets, we consider only links with an overall loss rate of less than 20% (these links are of most use to higher layers).

First, note that carrier-sensing senders make no transmissions above the diagonal line $\rho = \sigma$ in Figure 2-6. When we ignore the carrier sense line (Figure 2-7), we see that senders can achieve high link qualities above the diagonal $\rho = \sigma$. This suggests that the energy detect method of carrier sense is forgoing some good transmission opportunities, an observation shared by others [14, 24]. In Chapter 6 we discuss some ways of capitalizing on these transmission opportunities.

2.5.4 Abandoning carrier sense

Figure 2-8 shows the distribution of achieved throughput over all links in the sensor network. At 1 and 4 pps, enabling per-packet carrier sense results in greater throughput than disabling it altogether. This is because while per-

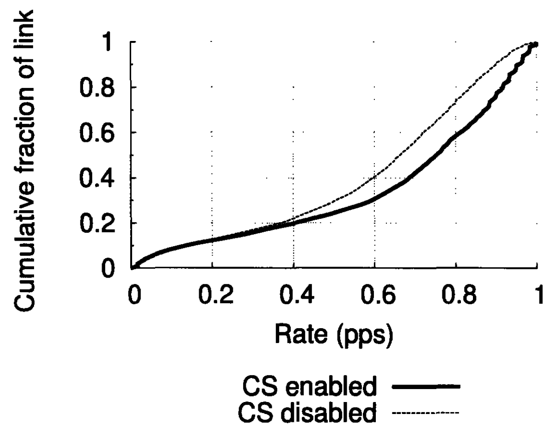


FIGURE 2-8—The achieved link throughput distribution in the sensor network when each node offers 1 packet/s.

packet carrier sense may slow down an individual transmission, our experiments take this into account and transmit such that the nominal transmission rate is equal to the actual transmission rate. At 8 pps, however, some carrier-sensing nodes cannot keep up with the offered load, because they spend too much time deferring. As a result, throughput suffers and consequently, nodes achieve higher throughput with carrier sense disabled. Thus even though carrier sense improves link quality at high loads, under extremely high loads, the improvement in link quality might not be worth the time it takes in deferral.

★ ★ ★

Sensor networks often exhibit a "collection" traffic pattern where traffic in the network funnels in to an access point. We now take a look at the problem of wireless loss in the context of sensor networks.

2.6 WIRELESS SENSORNET "CONGESTION COLLAPSE"

Provisioning a wireless sensor network so that congestion is a rare event is extremely difficult. Sensor networks can deliver myriad types of traffic, from simple periodic reports to unpredictable bursts of messages triggered by external events that are being sensed. Even under a known, periodic traffic pattern and a simple network topology, congestion occurs in wireless sensor networks because radio channels vary in time (often dramatically)

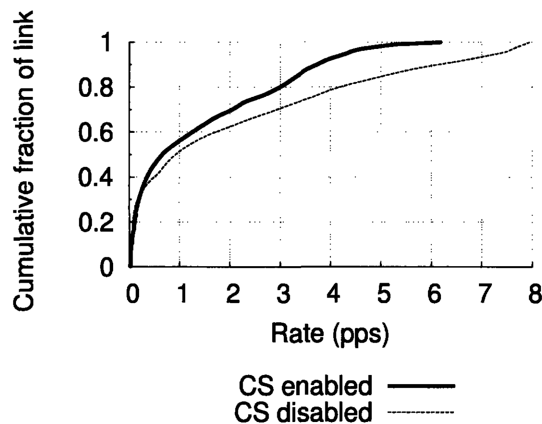


FIGURE 2-9—The achieved link throughput distribution in the sensor network when each node offers 8 packets/s. Under extreme load, disabling carrier sense improves throughput.

and concurrent data transmissions over different radio "links" interact with each other, causing channel quality to depend not just on noise but also on traffic densities. Moreover, the addition or removal of sensors, or a change in the report rate can cause previously uncongested parts of the network to become under-provisioned and congested. Last but not least, when sensed events cause bursts of messages, congestion becomes even more likely.

In wired networks and cellular wireless networks, buffer drops and increased delays are the symptoms of congestion. Over the past many years, researchers have developed a combination of end-to-end rate (window) adaptation and network-layer dropping or signaling techniques to ensure that such networks can operate without collapsing from congestion. In addition to buffer overflows, a key symptom of congestion in wireless sensor networks is a degradation in the quality of the radio channel caused by an increase in the amount of traffic being sent in *other* parts of the network. Because radio "links" are not shielded from each other in the same way that wires or provisioned cellular wireless links are, traffic traversing any given part of the network has a deleterious impact on channel quality and loss rates in other parts of the network. Poor and time-varying channel quality, asymmetric communication channels, and hidden terminals all make even well-regulated traffic hard to deliver. In addition, under traffic load, multi-hop wireless sensor networks tend to severely penalize packets that traverse a larger number of radio hops, leading to large degrees of unfairness.

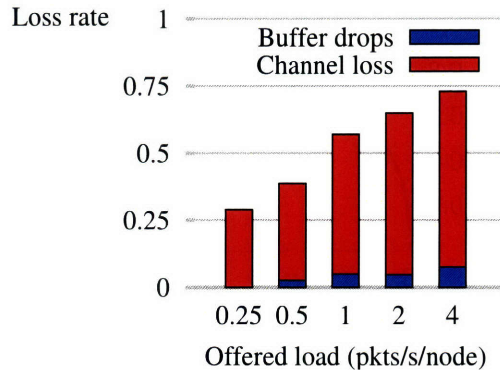


FIGURE 2-10—Congestion collapse in a testbed deployment with no congestion control strategy. Channel and buffer loss rate as a function of per-node offered load.

2.6.1 Metrics for sensornet wireless loss

This section diagnoses the two key symptoms of congestion collapse in wireless sensor networks. The following results are derived from our Mica2 wireless sensor network testbed, described in Section 2.5.1. Every node generates data at a constant rate, which other sensors forward over a multihop network to a single sink. As the offered load increases, loss rates quickly increase. Figure 2-10 shows network-wide loss rates for various offered loads, separating losses due to wireless channel errors from losses caused by a lack of buffer space. We see that channel losses dominate buffer drops and increase quickly with offered load. This dramatic increase in loss rates is one of the two symptoms of congestion collapse.

The second symptom of congestion collapse is starvation of most of the network due to traffic from nodes one hop away from the sink. Figure 2-11 illustrates this phenomenon. Given a percentage of packets p received from a given node at the sink, the complementary CDF plots the fraction of sensors that deliver at least p percent of their data to the sink. We see that as the offered load increases, a decreasing number of nodes get a disproportionately large portion of bandwidth.

Congestion collapse has dire consequences for energy efficiency in sensor networks, as Figure 2-12 shows. When offered load increases past the point of congestion, fewer bits can be sent with the same amount of energy. The network wastes energy transmitting bits from the edge towards the sink, only to be dropped. We call this phenomenon *livelock*.

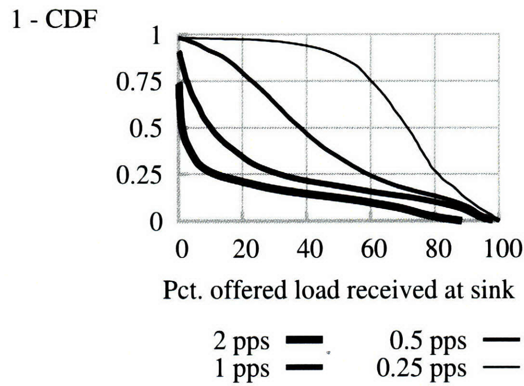


FIGURE 2-11—Congestion collapse in a testbed deployment with no congestion control strategy. Percentage of each node's offered load that is received at the sink (complementary CDF).

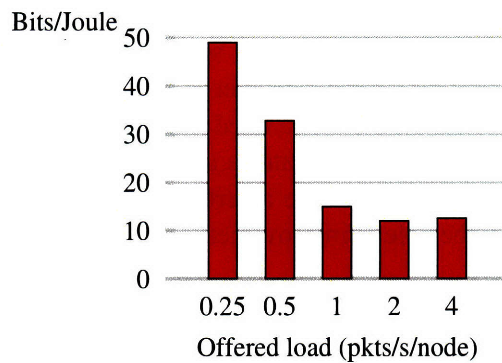


FIGURE 2-12—Congestion collapse in a testbed deployment with no congestion control strategy. Network-wide bits successfully transmitted per unit energy.

2.7 RELATED WORK

Capacity of multihop networks. In a seminal paper [42], Gupta and Kumar show that in an n-node multihop wireless network with a SINR-based interference model, throughput available to each node scales as $\Theta\left(\frac{1}{\sqrt{n}}\right)$ under optimal circumstances. Their work suggests that it may be advantageous to favor localized traffic patterns, rather than traffic traversing a larger portion of the network. This underscores the importance of traffic pattern, as well as medium sharing, in the performance of the network.

In a small-scale 802.11 testbed, Li et al. observe multihop interference which limits the end-to-end throughput a short forwarding chain of 802.11 nodes can achieve [69]. They note that because the radio causes some degree of interference to receivers whose packets it cannot detect via carrier sense, using carrier sense does not solve the problem. Padhye et al. make similar observations of pairwise interference between links, in an empirical study of interference in an 802.11 testbed [57].

More recently, Yang and Vaidya [132] examine the choice of the carrier sense range and its effect on capacity, taking into account MAC layer overhead. We share their observations about the impact that carrier sense and varying traffic loads have on overall network capacity. We propose to experimentally evaluate the simulation results in their work at both large and small scales.

Wireless sensornet congestion. As part of CODA [124], Wan et al. propose to detect congestion in a wireless sensor network using a technique called *channel sampling*. When a packet is waiting to be sent, the sensor samples the state of the channel at a fixed interval. Based on the number of times the channel is busy, it calculates a utilization factor. The node infers congestion if utilization rises above a certain level. We speculate that congestion control may be able to improve network capacity when carrier sense fails.

Medium access control. Whitehouse et al. present a collision detection and recovery technique for wireless networks that takes advantage of the capture effect [127]. Their technique detects packet collisions by searching for a preamble sequence throughout packet reception and recovers by re-synchronizing to the last detected preamble. We note that capture is a phenomenon that few protocols make explicit use of and that it might be exploited to make a more informed channel access decision.

Calì et al. analyze CSMA from the standpoint of throughput. They propose replacing the uniform-distribution contention window of 802.11 with a p-persistent backoff protocol [13]. By estimating the population size, they choose p to maximize system throughput when *all* nodes *always* have a packet ready for transmission. They show that 802.11 yields suboptimal throughput under this workload, and that their algorithm can approach optimal throughput under the same conditions.

Tree-splitting collision resolution [9,36,35] resolves collisions after they occur. Tree-splitting schemes require nodes participating in the contention-resolution protocol to assume that transmissions sent to other nodes in the same phase of contention resolution were successfully received. Since interference is a property of the receiver, this assumption is not always true, especially in a multihop network. Tree-splitting schemes also incur the performance penalty of a probable collision the first time many nodes become backlogged.

There are also proposals to use receiver-based feedback of channel conditions in making transmission decisions to improve the performance of CSMA. E-CSMA [24] uses observed channel conditions at the transmitter (RSSI, for example), and receiver-based packet success feedback to build a per-receiver probability distribution of transmission success, conditioned on the channel conditions at the sender at the time of transmission. Then a node makes a transmit/defer decision based on transmitter channel conditions just before sending a packet.

Tuning the carrier sense threshold. Fuemmeler et al. [28] study the choice of carrier sense threshold (defined in Section 2.4) and transmit power for 802.11 mesh networks. They conclude that senders should keep the product of their transmit power and carrier sense threshold equal to a fixed constant.

Zhu et al. present an analytical model for deriving an optimal carrier sense threshold [134], but their model does not take into account MAC layer overhead. The authors also propose a distributed algorithm that adapts the carrier sense threshold of an 802.11 mesh network, and present simulation results validating its efficacy.

Desilva et al. [21] found that carrier sense can unnecessarily suppress an 802.11 receiver from responding to RTS messages. They observe that a successful reception of a RTS message is a good indication that subsequent transmissions from the RTS sender can overcome the current noise levels observed at the receiver, even when the noise level is within carrier sensing

range. To increase efficiency, they propose 802.11 receivers use a different threshold for carrier sense prior to transmitting a CTS message.

Multiuser detection and interference cancellation. Conventional receivers, which are the focus of this dissertation, treat transmissions other than the one that they intend to receive as noise, so henceforth we refer to the signal to noise plus interference ratio (SINR). It is possible, however, to treat transmissions other than the one the receiver desires to decode not as noise, but as transmissions [97]. To that end, Verdú showed that the maximum-likelihood K-user sequence detector consists of a bank of per-user matched filters followed by a Viterbi algorithm that takes all users into account [122]. Unfortunately the maximum likelihood K-user receiver has high complexity, making it impractical. Subsequently, many suboptimal but practical interference cancellation schemes have been proposed for CDMA systems (see [125], for example), but commodity wireless local area networking hardware continues to use conventional receivers.

★ ★ ★

With the goal of maximizing network capacity, we have highlighted the existing problems of carrier sense. We have presented experimental results showing inefficiencies in two separate network testbeds, the first a large-scale deployment of narrowband FM radio sensor nodes, and the second a small-scale deployment of spread spectrum 802.11 radios.

In the next chapter we introduce our approach to the problems outlined so far, a new interface to the physical layer that sheds the wired abstraction, enabling higher-layer protocols to work at the sub-packet granularity.

3

The SoftPHY Physical-Layer Interface

WE NOW INTRODUCE a new physical layer interface called the *SoftPHY* interface. The SoftPHY interface is an extension to the physical layer that allows more information to flow upwards while maintaining the "digital abstraction" which divides the physical layer from higher layers. In Chapters 5 and 6 respectively, we use the SoftPHY interface as a key building block upon which we address the problems introduced in Chapter 1, and many other problems in wireless networking.

Chapter overview. In this chapter, we begin with a short summary of status quo problems, then present the SoftPHY interface and discuss how it fits into the layered architecture of a wireless networking stack. Then in Section 3.2, we go into significant technical detail as we describe our Zigbee and 802.11-like implementations of the SoftPHY interface. In the process, we describe how coded and uncoded communications systems in general (including those using cutting-edge concatenated and soft decision decoders) may implement the SoftPHY interface to realize gains in performance. We finish the chapter with experimental results from the aforementioned systems showing the general utility of SoftPHY hints, including SoftPHY hints in a communications system with structure identical to the IEEE 802.11a [52] physical layer.

Précis of status quo problems. In many current data communications systems [50, 52, 51, 53, 54], the physical layer outputs only a packet containing a sequence of bits after demodulation and channel decoding: we say

it presents a *digital abstraction* to higher layers. Additionally, the physical layer or layers close to it checksum the packet, discarding packets with any bit errors. This wired abstraction is rather limiting, since even after applying substantial channel coding, higher layers frequently lose packets, especially in the case of packet collisions.

3.1 DESIGN OVERVIEW

The SoftPHY interface consists of the digital abstraction with two simple modifications. In the following discussion, we will denote the granularity in bits with which the physical layer makes codeword or symbol decisions as b.[1] In increments of b bits, the physical layer assembles all the received bits, and passes the frame through the wired abstraction and up to the link layer. The SoftPHY interface modifies this interface in the following two ways.

1. The SoftPHY interface removes any checksum-based packet filtering that the physical and/or link layers may have implemented, effectively removing the wired abstraction described in Chapter 1.
2. For each group of b bits passed up, the SoftPHY interface also passes up a measure of confidence in the correctness of those bits on a standard scale.[2] We call this confidence the *SoftPHY hint* associated with those b bits decoded.

The details of how the physical layer calculates the SoftPHY hint depend on the modulation and coding in use. However, most demodulators and decoders can be easily modified to maintain this information, as we show in the next section.

Maintaining the digital abstraction. One benefit of the status quo layered receiver architecture is that the physical layer provides a digital abstraction, isolating layers above it from underlying implementation complexity, and allowing either the physical layer or the networking stack above it to be modified without changing the other. While a variety of physical layer implementations can provide the SoftPHY interface, the semantics of SoftPHY hints are tied to the details of the physical layer, potentially violating the digital abstraction.

[1]This quantity is easily identified in most receivers, and in particular, in each of the receivers we discuss in Section 3.2.

[2]This measure of confidence is called *side information* or *soft information* in the literature (see esp. [6, 97]).

However, layers above the SoftPHY interface are not aware of how SoftPHY hints are calculated: the hints are computed on a scale that is standardized across different physical layers. Thus, while SoftPHY hints themselves are PHY-*dependent*, layers above the physical layer use SoftPHY hints in a PHY-*independent* manner, retaining the benefits of the physical layer's digital abstraction.

SoftPHY and coding. One way that a communications system can add redundancy is to use a modulation with a large separation between constellation points relative to the amount of noise in the channel. Another way a communications system can add redundancy is to apply channel coding to the communications channel. This usually involves *encoding* the information at the transmitter before modulating it, and *decoding* the demodulated information at the receiver after demodulating it, both of which we describe below in Sections 3.2.2 and 3.2.3. Independent of the way (via modulation or via coding) that the communications system introduces redundancy, we show in this chapter that SoftPHY hints can leverage this redundancy for performance gains.

Independent of the coding in use, the decoder may either use "hard" symbol decisions $\hat{\mathbf{a}}_\mathbf{i}$ or the "soft" symbol samples $\mathbf{y}_\mathbf{k}$ from the demodulator (see Figure 3-1 on page 55). The former case is called *hard decision decoding* (HDD), and the later is called *soft decision decoding* (SDD). Whichever choice the receiver makes with regard to hard or soft decision decoding, the communications system retains a large coding gain whose magnitude depends on the amount of redundancy the code adds. For bit error rates between 10^{-6} and 10^{-2} (the typical operating points of the communications systems we are concerned with), SDD yields the same bit error rate as HDD, at an SINR lower by approximately 2.5 dB [97]. Independent of the SDD versus HDD design choice at the decoder, we show in this chapter that SoftPHY hints can leverage the redundancy inherent in a channel code.

End-to-end principles in SoftPHY. While the physical layer generates SoftPHY confidences to each group of b bits, it performs no further processing on the SoftPHY values. Higher layers thus retain the flexibility to choose how to interpret the SoftPHY hint associated with each b-bit group. This design choice reflects an application of the end-to-end principle in systems design [103]. Higher layers would not retain the flexibility they need, for example, in a design which quantized the SoftPHY hints into two levels, "good" and "bad," before passing them up to higher layers.

In addition, the present design allows higher layers to adapt their decisions on how to handle each bit based on multiple observations. For example, higher layers may compute the variance of SoftPHY hints over a time window, and use this effective measure of SINR to adapt their decisions [64, 131]. Or, as in Chapter 5, higher layers could threshold the SoftPHY hints, quantize them into two levels ("good" and "bad"), and then observe the correlation between a particular threshold and the correctness of the hint, adapting the threshold dynamically.

★ ★ ★

We now examine the physical layer, describing in detail how to modify three common receiver designs to pass SoftPHY hints up the network stack.

3.2 PHYSICAL-LAYER DESIGN

The first receiver design we consider is a physical layer without any coding layered above the demodulator. Then in the following two sections, we show how SoftPHY hints can be incorporated into systems that add redundancy through coding. Section 3.2.2 shows how to implement SoftPHY hints in a system with block coding; one important case of this is the Zigbee physical layer implementation we used to evaluate our system for reliable retransmissions in Chapter 5. In Section 3.2.3, we describe how SoftPHY works in a system with convolutional coding.

3.2.1 Uncoded communications

We begin by examining the digital receiver shown in Figure 3-1. This simple receiver design forms the conceptual basis upon which more sophisticated receivers build. The modulation we will consider in this section is *memoryless*, meaning that the information sent in the *n*th signaling interval is independent of the information sent in all other intervals.[3]

In this memoryless, uncoded communications system, we send one of M signals $s_1(t), ..., s_M(t)$ in each signaling interval. Each symbol encodes $b = \log_2 M$ bits of information.

Figure 3-1 shows an uncoded digital receiver that has been augmented to return SoftPHY hints. The first step in processing the incoming signal $r(t)$ is

[3]More sophisticated modulations and the coding techniques we discuss later introduce memory; see Proakis [97] for more information.

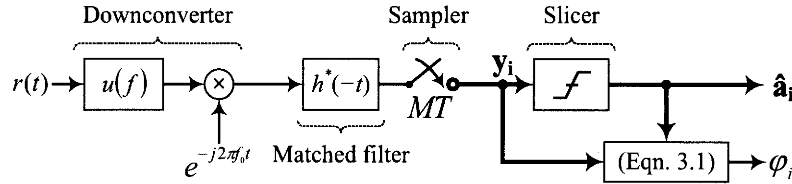


FIGURE 3-1—SoftPHY in an uncoded receiver. For each output symbol constellation point $\hat{\mathbf{a}}_\mathbf{i}$, the demodulator also produces a SoftPHY hint φ_k (defined in Equation 3.1) corresponding to the b bits encoded by $\hat{\mathbf{a}}_\mathbf{i}$.

to *downconvert* it from a signal modulated at a radio frequency (in the systems we are concerned with, typically tens of MHz to GHz) to a *baseband signal*. The first two blocks shown in Figure 3-1 accomplish this. The resulting signal is a complex-valued *baseband signal*. The structure that maximizes SNR at its output [6] is a filter $h^*(-t)$ matched to the time-reversed shape[4] of the transmitted signal as seen through the channel, $h(t)$. where M is the number of symbols in the signal constellation. After sampling the filtered signal, the key element in the receiver is the *slicer*, which quantizes the sampled signal to one of a few complex-valued *symbol* constellation points $\hat{a}_i$. For each quantized symbol constellation point, we obtain SoftPHY hints

$$\varphi_k = K_c \cdot \|\hat{\mathbf{a}}_\mathbf{i} - \mathbf{y}_\mathbf{i}\| . \tag{3.1}$$

This SoftPHY hint is the distance in signal space between the received constellation point and the decoded symbol's constellation point, scaled by a constant factor K_c that depends only on the modulation in use. This scaling factor standardizes the range of the SoftPHY hints φ_k across different modulations, so that higher layers need not concern themselves with the specifics of how the signal constellation is arranged.

3.2.2 Block-coded communications

One popular way of adding redundancy to a communications system is using *block coding*. The basic idea behind block coding is to use only a set of finite-length vectors to transmit; each one of these vectors is called a *codeword*, and the elements of each codeword are called *symbols*.[5] The common

[4]In this discussion we assume only one transmitted pulse shape, for simplicity.

[5]Codeword symbols are not to be confused with the channel symbols described in Section 3.2.1, although the two concepts frequently coincide.

case, which we consider in this section, occurs when there are two possible symbols, 0 and 1, in which case we are using a *binary* code.

In a binary block code with length-n codewords there are 2^n possible codewords. Of these 2^n possibilities, we select a subset of size $M = 2^k$ to form the *codebook* $\{\mathbf{C_1}, ..., \mathbf{C_M}\}$, a restricted set of codewords that the transmitter may send over the channel to the receiver. At the transmitter, the encoding process maps consecutive source data blocks of size k onto code data blocks of size n. We refer to the resulting code as an (n, k) binary block code of rate $R = k/n$. The sender then groups the code data into channel symbols encoding $k_s \geq 1$ bits each, and sends the channel symbols over the air, modulated over some baseband transmit pulse shape.

Decoding block codes. In a hard decision decoding design, the decoder uses the hard symbol decisions $\hat{\mathbf{a}}$ from the demodulator (see Figure 3-1 on page 55). The maximum-likelihood decision rule then becomes the following. For each codeword C_m in the codebook ($m = 1, ..., M$), the receiver computes the Hamming distance between the received symbols and C_m:

$$d_H(\hat{\mathbf{a}}, \mathbf{C_m}) = \sum_{i=1}^{n} \hat{a}_i \oplus C_{m,i} \tag{3.2}$$

where $C_{M,i}$ ($i = 1, ..., n$) is the ith symbol in the mth codeword. The receiver decides on codeword r where

$$r = \arg\min_{r} d_H(\hat{\mathbf{a}}, \mathbf{C_r}). \tag{3.3}$$

In the hard decision decoder, we define SoftPHY hint φ as

$$\varphi = K_c \cdot (d_{\min} - d_H(\hat{\mathbf{a}}, \mathbf{C_r})) \tag{3.4}$$

where $d_{\min}$ is the minimum Hamming distance between any two codewords, and K_c is a constant chosen to scale the SoftPHY hints to a standard scale. Both these constants are dependent on the block code being used.

In lieu of HDD, a decoder can use *soft decision decoding* (SDD) [6] which works directly on samples of received symbols y_k, before they are sliced, thus using more information to make its decisions. However, SDD will still produce incorrect codewords at very low SINR, and thus does not recover correct bits particularly well during packet collisions.

In a soft-decision decoding design, the decoder calculates the *correlation* C between samples of received symbols $\mathbf{y}$ and each codeword $\mathbf{C_i}$ (whose

jth constituent symbol is c_{ij}):[6]

$$C(\mathbf{y}, \mathbf{C_i}) = \sum_{j=1}^{n} z_j c_{ij}. \tag{3.5}$$

Like the maximum-likelihood HDD, the maximum-likelihood SDD receiver then decides on codeword r where

$$r = \arg\min_i C(\mathbf{y}, \mathbf{C_i}). \tag{3.6}$$

Then the SoftPHY hint φ is computed as

$$\varphi = K_c \cdot (d_{\min} - C(\mathbf{y}.\mathbf{C_r})) \tag{3.7}$$

Finally, we note that a hybrid approach is possible, where the receiver makes the HDD maximum-likelihood decoding decision in Equation 3.3, but computes the SoftPHY hint based on soft information, using Equation 3.7. This has the drawback of requiring more storage complexity in the receiver, since the demodulator cannot discard soft symbol information $\mathbf{y}$ immediately. We also note that HDD-based SoftPHY hints are tantamount to SDD-based SoftPHY hints followed by a quantization step. We have implemented both SDD and HDD-based SoftPHY hints, and find that in a large Zigbee testbed, HDD-based SoftPHY yield significant performance gains with a minimal overhead in receiver complexity, and have the advantage of being easily represented in four bits of information.

The preceding two communication models apply to spread spectrum radio, a popular technique used to increase resistance to narrowband noise, and decrease (but not eliminate) interference from other transmissions [6, 97, 109]. In particular, the preceding two models directly apply to both IEEE 802.15.4 [54] (Zigbee) and IEEE 802.11b [51] (WiFi), two common direct sequence spread spectrum radio standards, which we briefly discuss now.

IEEE 802.15.4 (Zigbee). Zigbee [54] uses a (32,4) binary block code, with a code rate $R = 1/8$, and MSK modulation[7] [97], so $k_s = 2$. In spread spectrum modulation, each coded data bit is called a *chip*, and the rate of data chips differs from the rate of data bits. In the Zigbee system, chips are

[6]We note the existence of algorithms that implement the maximum-likelihood correlation computation efficiently [128].

[7]MSK modulation is equivalent to offset QPSK (O-QPSK) modulation with half-sine pulse shaping [89].

sent at 2 Mchips/s, eight times faster than the uncoded data bits coming in to the system at 250 Kbits/s. This shorter chip duration results in a spreading of the data in frequency, hence the term spread spectrum. To compute Zigbee SoftPHY hints, we leverage the fact that any 32-bit chip sequence lies within a distance of 16 to a valid Zigbee codeword. We therefore set $K_c = 1$ in Equation 3.4, resulting in a SoftPHY hint ranging from zero (low confidence) to 15 (high confidence) that can be represented compactly with four bits of information.

We evaluate the SoftPHY hints from our Zigbee receiver implementation below in Section 3.3 and evaluate their performance in a reliable retransmission protocol in Chapter 5. The Zigbee receiver in those performance evaluations uses hard decision decoding and the Hamming distance-based SoftPHY hint. Koteng [67] investigates this and other Zigbee receiver designs in depth.

IEEE 802.11b (WiFi). At its lowest two rates, 1 Mbit/s and 2 Mbits/s, IEEE 802.11b [51] uses differential BPSK and differential QPSK modulations, respectively, at 1 Msymbol/s. The symbols are spread by an 11-chip Barker spreading sequence. At the receiver, the maximum-likelihood detector is a filter matched to the spread signal (a "chip-matched" filter), so the receiver design and SoftPHY hints are identical in structure to the uncoded receiver described above.

At 5.5 Mbits/s and 11 Mbits/s, IEEE 802.11b uses the same chipping rate of 11 Mchips/s but uses complementary code keying [90] (CCK) to code eight chips per symbol. CCK codes four or eight bits per symbol, for 5.5 Mbits/s and 11 Mbits/s respectively. Again, the maximum-likelihood detector is a chip-matched filter.

3.2.3 Convolutionally-coded communications

Another popular way of adding redundancy to a communications system is to employ *convolutional coding*. In this section we describe how convolutional codes work and how they are typically decoded using the well-known Viterbi algorithm. Then we will see how to use a slightly different decoding algorithm to derive SoftPHY hints for convolutionally-coded data.

Convolutional coding works by passing input data **A** through a linear shift register k bits at a time. The shift register contains $k \cdot (K - 1)$ registers,

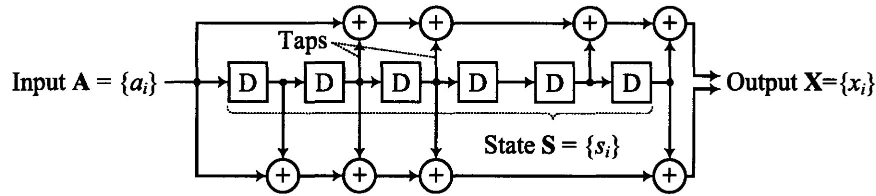


FIGURE 3-2—A convolutional encoder for a ($k = 1, n = 2, K = 7$) binary convolutional code. This convolutional coder generates the maximal free distance convolutional code with constraint length seven which IEEE 802.11a uses (see discussion on page 63).

whose contents at time i we denote s_i.[8] Some number of *taps* are attached to the inputs and various stages of the shift register, leading through combinatorial logic to the output of the encoder at time i, x_i.

The convolutional encoder may be viewed as a finite state machine with $2^{k\cdot(K-1)}$ possible states.[9] For every k bits a_i presented to the convolutional encoder at each clock cycle, n bits x_i are output, resulting in a convolutional code of rate $R = k/n$. Figure 3-2 shows an example where the shift register of $K = 7$ delay stages labeled "D" (including the input) clocks one bit into itself in each clock cycle ($k = 1$), resulting in two output bits per clock cycle ($n = 2$). Each of the $\oplus$ operators in the figure is a base-2 sum (exclusive or) operation, and we denote the evolution of the input, state, and output in time respectively as vectors **A**, **S**, and **X**.

This convolutional coding model is applicable to the IEEE 802.11a [52] OFDM physical layer, which we describe in detail below. It also applies to the most widespread IEEE 802.11g [53] OFDM physical layer, the "extended-rate PHY" (ERP).

The trellis diagram. We now introduce the key data structure used to reason about convolutional codes, the *trellis*. Each *stage* of the trellis (along the ordinate axis of Figure 3-3) consists of $2^{k\cdot(K-1)}$ states, each represented by a different point along the abscissa of the figure. Each of these states corresponds to a possible state that the convolutional coder at the transmitter may be in at any time. Originating from each state are 2^k *branches* representing all possible state transitions of the coder, given a particular input. In the trel-

[8]With the inclusion of the k current input bits, this results in kK bits of state at any instant in time.

[9]Since the shift register's succeeding state does not depend on the final k bits in the shift register.

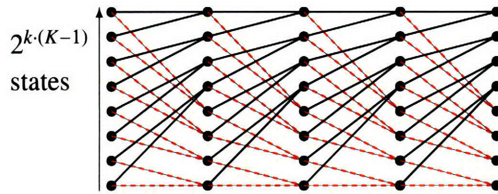


FIGURE 3-3—Trellis diagram for a ($k = 1$, $n = 2$, $K = 4$) binary convolutional code with maximal free distance $d_{free} = 6$.

lis of Figure 3-3, $k = 1$, so there are two branches originating from each node, the lower broken branch for a "0" bit input to the convolutional encoder, and the upper solid branch for a "1" bit input. Thus given an encoder initially in the all zero state, there are one-to-one correspondences between the encoder input data, the encoder's state transitions, and paths through the trellis diagram.

The key factor that determines a convolutional code's error correcting capability is the minimum Hamming distance between any two sequences of coded bits [97]. This distance is called the *free distance* of the code, and we denote it d_{free}. By computer search, $R = 1/2$ maximal free distance codes are well known for various constraint lengths up to $K = 14$ [97], and have identical trellis structure. In particular, the structure of the trellis in Figure 3-3 is identical to the structure of the maximal free distance convolutional code used in IEEE 802.11a with $K = 7$. We present the trellis for the $K = 4$ code here for clarity of exposition.

Decoding convolutional codes. The most popular way of decoding convolutional codes is the Viterbi algorithm, an excellent summary of which is given by Forney [27]. Given a number of observations made by a receiver, the Viterbi algorithm selects the most likely transmitted sequence. Following in part the discussion in Barry et al. [6], we summarize the Viterbi algorithm here in order to compare it with the BCJR algorithm. We use the later algorithm to generate SoftPHY hints for convolutional codes.

In our notation, we start with a vector of channel observations $\mathbf{Y}_0$, and our goal is to select the vector of convolutional encoder states $\mathbf{S}_0$ which maximizes the probability $\mathrm{P}_{\mathbf{S}|\mathbf{Y}}(\mathbf{S}_0|\mathbf{Y}_0)$.

For each branch of the trellis in Figure 3-3, we define a *branch metric from trellis state p to state q at stage i*:

$$\gamma_i(p,q) = f_{y_i|x_i}\left(y_i\left|x^{(p,q)}\right.\right) p_{a_i}\left(a^{(p,q)}\right), \tag{3.8}$$

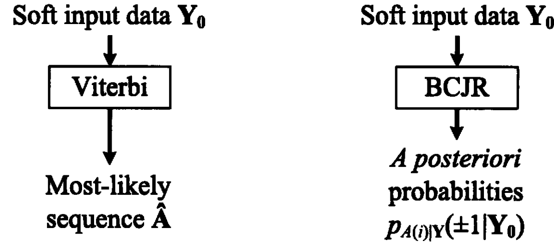


FIGURE 3-4—Comparison between the Viterbi and BCJR algorithms for maximum-likelihood sequence detection of binary convolutionally-encoded data: for each transmitted bit, the BCJR algorithm returns the *a posteriori* probabilities of whether a +1 bit or a −1 bit was transmitted.

where $x^{(p,q)}$ is the encoder output upon a $p \to q$ state transition. Given a vector of observations $\mathbf{Y}_0$, the *path metric for path* $\mathbf{S}_0$ is

$$\gamma(\mathbf{S}_0) = f_{\mathbf{Y}|\mathbf{S}}(\mathbf{Y}_0 \mid \mathbf{S}_0)\, p_{\mathbf{S}}(\mathbf{S}_0) = \prod_{i=0}^{L+K-1} \gamma_i(p_i, q_i), \tag{3.9}$$

where the path metric multiplies the branch metrics corresponding to the branches that form the path $\mathbf{S}_0$, i.e.: $\mathbf{S}_0 = \{p_0,\ p_1 = q_0,\ p_2 = q_1, \ldots, p_{L+K-2} = q_{L+K-1}\}$.

Define the *partial survivor path* for state p at time i as the partial path beginning at $i = 0$ and zero state and leading to state p at time i, with maximal path metric (we denote this metric as $\alpha_i(p)$.

The key insight of the Viterbi algorithm is that to find the path with the largest path metric, we need only remember the partial survivor path for each possible state p at a given trellis stage i, because if the path with the largest path metric did not contain the partial survivor path, we could replace the path up to stage i with the partial survivor path and obtain a path with larger metric, a contradiction. Thus we obtain the following Viterbi recursion for computing the maximal partial path metrics:

$$\alpha_{i+1}(q) = \max_p \{\alpha_i(p)\gamma_i(p,q)\}. \tag{3.10}$$

With this recursion we can make one pass through the trellis from right to left, computing path metrics and forming the most likely path through the trellis, which as noted above, has a 1-1 correspondence with the most likely sequence of input data at the transmitter.

The BCJR algorithm. Given the same set of observations at the receiver as the Viterbi algorithm, the BCJR algorithm for sequence detection (Bahl, Cocke, Jelinek, Raviv [5]) selects the sequence of most likely symbols that the transmitter sent. Here we use the algorithm to decode the output of a convolutional encoder, as observed across a noisy channel. We now summarize the BCJR algorithm and compare it to the Viterbi algorithm; again some parts of our exposition follow Barry et al. [6]. Then we show how we can process the output of the BCJR algorithm to calculate SoftPHY hints φ_i for the bits corresponding to each transmitted symbol.

The BCJR algorithm uses the same trellis structure introduced above, but as shown in Figure 3-4, instead of an estimate of the most likely transmitted data $\hat{\mathbf{A}}$, the BCJR algorithm outputs *a posteriori probabilities* (APPs) $p_{A(i)|\mathbf{Y}}(\pm 1|\mathbf{Y}_0)$ that give us a measure of the confidence the decoder has in each symbol.

Suppose we form a set $\mathcal{S}_a$ which contains all pairs of states (p, q) for which a trellis transition from state p to q corresponds to encoder input symbol a. Then we may express the desired APPs in terms of *a posteriori state transition probabilities*:

$$p_{A(i)|\mathbf{Y}}(\pm 1|\mathbf{Y}_0) = \sum_{(p,q)\in\mathcal{S}_a} Pr(s_i = p, s_{i+1} = q|\mathbf{y}). \tag{3.11}$$

We can then decompose the APP state transition probabilities into

$$\sigma_i(p, q) = \alpha_i(p) \cdot \gamma_i(p, q) \cdot \beta_{i+1}(q) \tag{3.12}$$

where $\gamma_i(p, q)$ is the Viterbi branch metric in Equation 3.8. It is also easy to show that the α and β metrics can be computed via the following recursions [6]:

$$\alpha_{i+1}(q) = \sum_{\text{all states } p} \alpha_i(p) \cdot \gamma_i(p, q) \tag{3.13}$$

$$\beta_i(p) = \sum_{\text{all states } q} \gamma_i(p, q) \cdot \beta_{i+1}(q) \tag{3.14}$$

The BCJR algorithm thus reduces to the following steps:

Step 1: Calculate the Viterbi branch metrics γ_i using Equation 3.8 on page 60.

Step 2: Calculate forward and reverse metrics α_i and β_i using Equations 3.13 and 3.14, respectively.

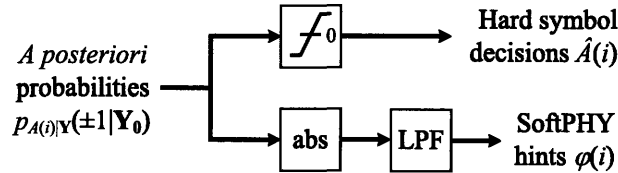


FIGURE 3-5—Postprocessing stages after the BCJR algorithm to generate data and SoftPHY hints from a posteriori probabilities.

Step 3: Calculate a posteriori state transition probabilities using Equation 3.12.

Step 4: Calculate APPs using Equation 3.11.

After we compute APPs using the BCJR algorithm, in the case of a binary convolutional code, we can obtain SoftPHY hints φ by calculating the log likelihood ratio of the two possible values of the encoder input a_i:

$$\varphi = \log\left(\frac{p_{A(i)|\mathbf{Y}}(+1|\mathbf{Y}_0)}{p_{A(i)|\mathbf{Y}}(-1|\mathbf{Y}_0)}\right) \tag{3.15}$$

Finally, we note that to achieve slightly reduced computational complexity, SoftPHY can use the output of the Soft-output Viterbi (SOVA) [44] algorithm, in the same manner as the BCJR output.

IEEE 802.11a WiFi. We have implemented the above channel coding techniques in a communications system for GNU Radio [39] that mirrors the design of IEEE 802.11a [52] WiFi except that it uses a smaller bandwidth for its transmission, due to bandwidth constrains imposed by the USRP [25] software defined radio.

802.11a uses the rate-1/2 convolutional encoder of constraint length $K = 7$ shown in Figure 3-2. To achieve different code rates, the coded data X is *punctured* at some rate R_p: a fraction R_p of coded bits are removed from the coded data stream according to a puncturing pattern agreed-upon a priori by the transmitter and receiver. Then the coded data stream is mapped to OFDM subcarriers, each of which operating using either BPSK, QPSK, QAM-16, or QAM-64 modulation. This results in another degree of freedom with respect to bit-rate control. The system interleaves the data onto non-adjacent (in frequency) OFDM subcarriers to guard against a frequency-selective fade causing adjacent subcarriers to fade simultaneously and cause

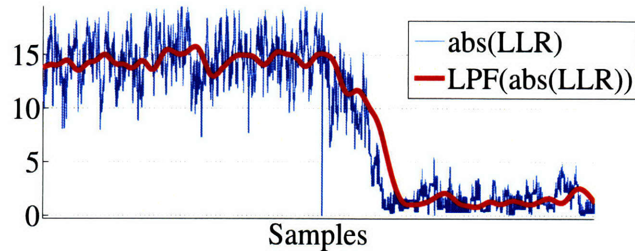


FIGURE 3-6—BCJR-based SoftPHY confidence in 802.11a over a packet during a packet collision over the later half of the packet.

runs of bit errors. A collision, however, still causes interference on all subcarriers, as we will illustrate below.

Our implementation consists of code written by the author and others. At the receiver, we have incorporated BCJR code from the GNU Radio codebase by Anastasopoulos, and have incorporated and contributed to the functionality of the OFDM GNU Radio codebase by Rondeau et al.. To guard against header and preamble corruption, especially using the techniques described in Chapters 5 and 6 that recover parts of packets, 802.11a switches to the lowest 802.11a rate (BPSK with a $R = 1/2$ code) for header and trailer transmission; we have implemented this functionality, writing new code to implement rate-switching within a single transmission. We have also written original code for the computation of SoftPHY hints as described above. Finally, we have contributed code to adapt Anastasopoulos's BCJR implementation to packetized data at changing modulation and coding rates, as demanded by the 802.11a specification.

In Figure 3-6, we show the SoftPHY hints φ our code generates over a single packet transmission. In practice, we expect that the SoftPHY hint would be passed through a low-pass filter to compensate for short bursts of noise and fast channel fading; the smooth curve in Figure 3-6 shows the result. From both curves, we can clearly discern that the first half of the packet is received with high confidence and the last half with low confidence. In fact, this is a picture of a packet collision where the colliding packet overlaps with the last half of the received packet, and the SoftPHY hints accurately reflect this situation.

FIGURE 3-7—Experimental Zigbee testbed layout: there are 31 nodes in total, spread over 11 rooms in an indoor office environment. Each unnumbered dot indicates one of 25 Zigbee nodes. The six software defined radio nodes are shown dark, labeled with numbers.

3.3 EVALUATION OF SOFTPHY HINTS

We now evaluate the predictive power of SoftPHY hints in two demanding but very different environments. First, we evaluate how well SoftPHY hints work in a multihop Zigbee network deployed over an entire floor of a building, and second, we evaluate SoftPHY in a small network over a quiet frequency band, at marginal signal to noise ratio. In both circumstances, we next show that SoftPHY hints are a good predictor of received bits' correctness. We leave an end-to-end performance evaluation of a SoftPHY-based protocol to Chapter 5.

3.3.1 SoftPHY hints in a busy network

We perform this experiment in the 31-node combined Zigbee/software radio testbed shown in Figure 3-7. Thus, transmissions in this experiment must contend with a large amount of interfering traffic from both the network itself, and from background 802.11 traffic.

Implementation. Each Zigbee sender is a telos mote with a Chipcon CC-2420 radio [117]. Senders run TinyOS[10] on the telos's TI MSP430 microprocessor. The CC2420 radio is a 2.4 GHz single-chip RF transceiver that uses direct-sequence spread spectrum (DSSS) at a bit rate of 250 Kbits/s as described in Section 3.2.2 on page 57.

[10]See `http://tinyos.net`.

Each of the Zigbee receivers is a computer connected to a software defined radio. The hardware portion of the receiver is a Universal Software Radio Peripheral (USRP) [25] with a 2.4 GHz daughterboard; the remainder of the receiver's functionality (demodulation and block decoding as described in Section 3.2.2) is implemented in software. The DSSS despreading function is approximately 1,500 lines of original code written in C++ in the GNURadio [39] framework by the author, with parts derived from code by Schmid [106, 107].

Experimental design. In our testbed, we have deployed 25 sender nodes over eleven rooms in an indoor office environment. We also deployed six receivers among the senders; in the absence of any other traffic, each receiver can hear between four and ten sender nodes, with the best links having near perfect delivery rates. All 25 senders transmit packets containing a known test pattern, at a constant rate. For each Zigbee codeword received, we measure the SoftPHY hint φ associated with that codeword using the computation in Equation 3.4.

Our experimental results below summarize data from Zigbee channel 11 at 2.405 GHz. Zigbee channel 11 overlaps with IEEE 802.11b channel 1, which carries active WiFi traffic in our building. Thus the experimental results we report next were obtained in the presence of significant background traffic. In addition, we have validated our experimental results on Zigbee channel 26, which has no overlap with 802.11 channels. We used GNU Radio tools [39] to verify that there was indeed a high level of background traffic on Zigbee channel 11 and indeed significantly less background traffic on Zigbee channel 26.

Results. Figure 3-8 shows the distribution of Hamming distance across each received codeword, separated by whether the codeword is correctly or incorrectly received (we know this because packet payloads contain a known test pattern). In the figure, data points represent averages of 14 runs and all error bars indicate 95% confidence intervals, unless otherwise indicated. Conditioned on a correct decoding, only about one in 100 codewords have a Hamming distance of two or more. Conversely, fewer than one in 10 incorrect codewords have a distance of two or less.

This result shows that one way higher layers can interpret this SoftPHY hint is by implementing a threshold test [65]. We denote the chosen threshold η, so that the higher layer labels groups of bits with SoftPHY hint $\varphi > \eta$ "good" and groups of bits with $\varphi \leq \eta$ "bad." Under this interpretation, the

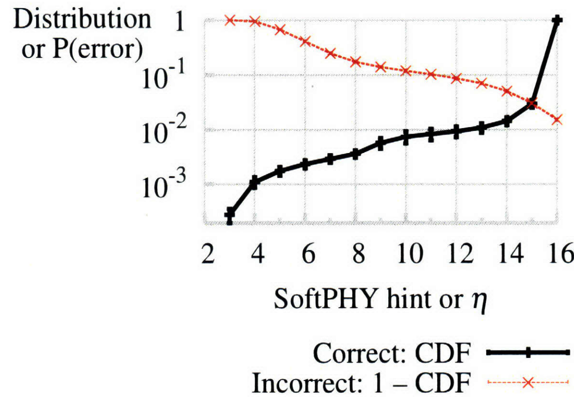


FIGURE 3-8—Distributions of Hamming distances for every codeword in every received packet, separated by whether the received codeword was correctly or incorrectly decoded. Under the threshold rule, these CDFs can be reinterpreted as curves plotting probability of misclassification.

curves in Figure 3-8 also give the probabilities of error for both types of mistakes, for the following reason. Consider the correct cumulative distribution function shown in the figure. Given a particular choice of η, the correct codeword CDF gives the fraction of codewords less than or equal to η. These are the codewords that the classification rule will label "bad," and thus the CDF yields the probability of error. A similar, converse argument shows that the complementary CDF yields the probability of mislabeling an incorrect codeword "good." Under the threshold rule then, the two curves in Figure 3-8 also show the probability of misclassification for correct and incorrect symbols, respectively.

We call the fraction of incorrect codewords that are labeled "good" the *miss rate* at threshold η. We see from Figure 3-8 that the miss rate is one in ten codewords at $\eta = 13$, initially a cause for concern. The saving grace is that when misses occur, it is highly likely that there are correctly-labeled codewords around the miss, and so PP-ARQ will choose to retransmit the missed codewords. Figure 3-9 verifies this intuition, showing the complementary CDF of contiguous miss lengths at various thresholds η. We see that a significant fraction of misses are of length one, and that long runs of misses are extremely rare.

Finally, we note that the result presented in this section are invariant across experimental runs that disable carrier sense, change the offered traffic load to be higher or lower, or change the background traffic to be absent.

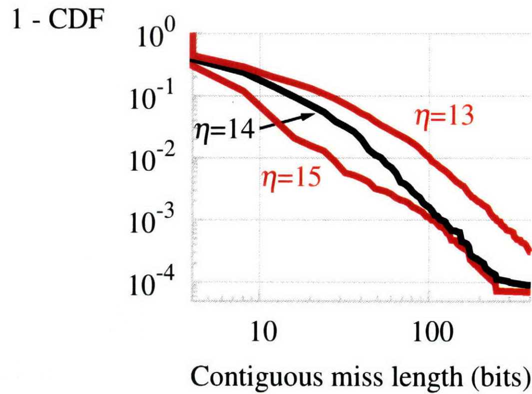


FIGURE 3-9—The distribution of lengths of contiguous misses in every received packet for various thresholds η. Most misses are short, only a few bits length.

3.3.2 SoftPHY hints at marginal SINR

We now turn from networks with significant interfering transmissions to an evaluation of SoftPHY hints in a quiet wireless channel at marginal SINR. We have implemented the design in Figure 3-1 for a DQPSK receiver. In this section, we evaluate these SoftPHY hints at marginal SINR. This represents the worst possible case for SoftPHY, because SoftPHY leverages redundancy available in the communication channel.

Implementation. We have implemented a software-defined radio transmitter and receiver using a combination of the USRP hardware with CppSim [91] and Matlab. The transmitter uses grey-coded QPSK with square-root raised cosine pulse shaping, for an aggregate data rate of 1.33 MBps. The receiver synchronizes on the incoming signal, compensating for carrier frequency offset, and then demodulates the signal, using differential detection of the QPSK modulation. The receiver computes uncoded SoftPHY hints by using the received phase of the signal θ_r. Since $\sin\theta \approx \theta$ for small θ, we approximate the second factor in Equation 3.1 with $|\theta_r - \angle\hat{a}_i|$ and set $K_c = 4/\pi$ to normalize the uncoded SoftPHY hints to the same $[0, 15]$ scale as the Zigbee hints. There is no channel coding layer in this radio.

Experimental design. To perform these experiments, we utilized a frequency band that does not overlap with 802.11 [113], the dominant source

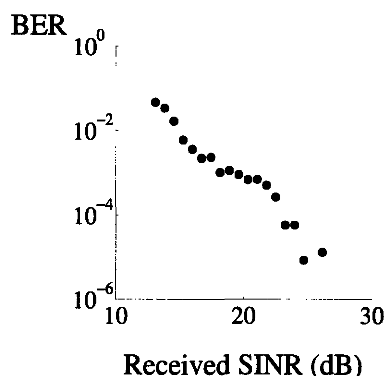


FIGURE 3-10—Bit error rate (BER) v. received signal to noise ratio for a DQPSK receiver in a "quiet" network.

of RF interference in our environment.[11] The experiments in this section use a software radio-based DQPSK transmitter and receiver pair, whose implementation is described above in Section 5.4.

To simulate links with varying amounts of path loss, we send a stream of packets between the two radios, modulating the transmit power of the stream of packets (we hold transmit power constant for the duration of each packet). At the receiver, we calculate the average received SINR for each packet and check the correctness of each bit in the packet. We also compute the SoftPHY hint for each symbol using Equation 3.1.

Results. Figure 3-10 shows the BER-SINR curve for the experiment. We note the high BER for relatively-high SINR, hypothesizing that better clock-recovery algorithms and of course coding would shift the curve left as is commonly seen in commercial radio receivers.

We partition the data into "good," "mid," and "bad" transmissions according to average SINR, as in Table 3.1. Figure 3-11 shows the cumulative distribution of SoftPHY hints in each regime. We see that SoftPHY hints are a good predictor of symbol correctness in the good regime, but an increasingly poorer predictor of symbol correctness as SINR decreases, as expected. We note that the SoftPHY hint φ we use here, per-symbol angular difference from the hard decision, is based on an uncoded modulation considering each symbol independently, and we have achieved significantly

[11]We used GNURadio tools to check for significant interference in our channel between runs of these experiments.

Label	SINR	BER
Good	SINR ≥ 21	BER ≤ 10^{-3}
Grey-zone	13 < SINR < 21	10^{-3} < BER < 10^{-2}
Bad	SINR ≤ 13	BER ≥ 10^{-2}

TABLE 3.1—Experimental regimes for evaluating SoftPHY at marginal SINR.

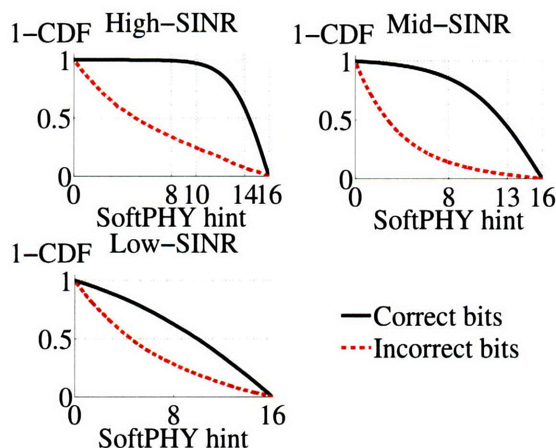


FIGURE 3-11—SoftPHY hints at marginal SINR in a quiet network. We present three sets of data corresponding to each of the three regimes in Table 3.1.

better results in interference-dominated experiments with coded modulations (see Figure 3-8).

These results (in the "bad" regime) illustrate the worst-case scenario for SoftPHY, a low SINR transmission in a relatively static (little interfering transmissions) channel. Under these conditions, we see the diminishing utility of SoftPHY hints. However, most mesh network traffic loads are the opposite: highly varying SINR due to packet collisions. Finally, we note that these results show a proof-of-concept in an alternative modulation.

★ ★ ★

In this chapter we have presented the SoftPHY interface, discussed several important design considerations, and described several physical layer design modifications for its implementation. One of the most important contributions of the SoftPHY interface is the myriad ways it can be used. To that end, in Chapters 5 and Chapter 6 we investigate several different uses

for the SoftPHY interface, starting in Chapter 5 with an enhanced protocol for reliable data retransmission.

4

Postamble-based Synchronization

IN MOST communications system designs, proper synchronization between transmitter and receiver is necessary before communication can proceed. In the systems we are interested in, synchronization occurs using a *preamble* attached to the beginning of every frame transmitted on the air. The preamble contains known data that assists the receiver's synchronization algorithm. The main contribution of this chapter is a *postamble decoding* mechanism to improve this process in the face of interference in a busy wireless network.

In Chapters 5 and 6 we use postamble decoding as the second key building block (in conjunction with the SoftPHY interface of Chapter 3) upon which we address the problems introduced in Chapter 1.

Chapter overview. We begin with a brief introduction to the topic of synchronization in digital receivers, describing three state-of-the-art synchronization algorithms that we have implemented for Zigbee and/or 802.11a. We then explain the need for a postamble, and describe the design of postamble-based synchronization and decoding in the context of each synchronization algorithm. Finally, we present experimental data for each of the three systems.

4.1 SYNCHRONIZATION IN DIGITAL COMMUNICATIONS

Synchronization is the process by which the receiver estimates and tracks the frequency and phase of two unknown properties of the radio frequency-

modulated signal that the transmitter sends over the air. From the perspective of the receiver, the first unknown is the transmitter's *carrier signal*, and the second unknown is the transmitter's *symbol timing clock*. Despite the fact that both properties are nominally known in any communications system, the receiver does not have precise knowledge of the frequency or phase of either signal.

Three key design choices. The following three design choices set the context for our discussion.

1. The synchronization designs we consider are frame-based (in contrast to radio systems that use time-continuous transmissions). The choice of frame-based transmissions follows from our choice of carrier-sense multiple access to share the wireless medium in time.

2. Each frame in our system includes a short[1] sequence of known data prepended to each frame, called a *training sequence*. The receiver can then use its shared knowledge of this data to detect the presence of the frame.

3. Our synchronization designs are implemented in all-digital logic (in contrast to a receiver implemented using analog components). This design choice reflects a trend (which started in the 1990s and possibly earlier) towards shifting more functions from analog signal processing to digital signal processing. This choice reduces the complexity of filters in the analog receiver front-end design [79].

4.1.1 Symbol timing recovery

In this chapter, we use the term "symbol" to refer to physical-layer symbols, the units by which the modulation (operating below any channel coding layer) makes decisions. A receiver needs to perform symbol timing clock recovery to determine when (i.e., with which frequency and phase) to sample each symbol in the incoming signal such that the probability of correct detection is maximized.

The challenge in recovering the frequency and phase of the symbol timing clock lies in the problem of mapping the samples taken with the receiver's symbol sampling clock (shown in the upper half of Figure 4-1)

[1]Typically on the order of 1% of the frame size.

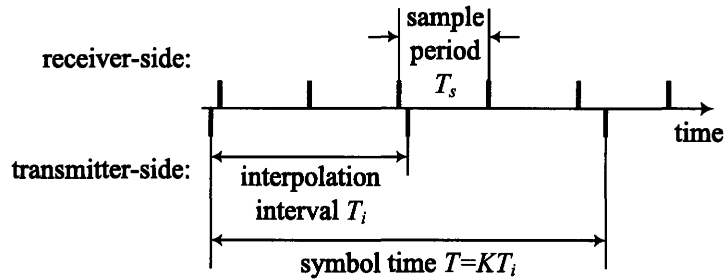


FIGURE 4-1—(Gardner [37]) To synchronize its symbol timing clock, the receiver must map the samples it has taken with its sampling clock (with period T_s, shown in the upper half of the figure) to the transmitter's symbol timing clock (with period T, shown in the lower half of the figure), optionally upsampling the symbols by a small integer factor K to form the interpolated symbol clock with period T_i.

to the transmitter's symbol timing clock (shown in the lower half of Figure 4-1). The majority of the literature does not present the problem in this way; two notable exceptions are Gardner's tutorial paper [37] and the texts by Meyr et al. [79] and Mengali and D'Andrea [75]. The designs we present in the next section take the foregoing considerations into account.

If the receiver is performing *coherent* detection, it also needs to perform carrier frequency recovery to estimate the incoming carrier signal's time-varying frequency and phase.

4.2 IMPLEMENTATION

We have implemented three distinct synchronization algorithms for each of the two standards (Zigbee and 802.11a) we have targeted.

4.2.1 802.11a (OFDM) synchronization

In this section, we describe the synchronization algorithm design for the 802.11a-like receiver we presented in Section 3.2.3. In brief, this receiver emulates the structure of the 802.11a system: variable-rate convolutional coding over OFDM modulation, except with a smaller signal bandwidth due to USRP [25] data throughput constraints.

The OFDM synchronization algorithm we use is due to Schmidl and Cox [108]; Ettus, Rondeau, and McGwier have implemented it and incorporated it into the existing GNU Radio code base [39]. As part of our 802.11a-like receiver design described in Section 3.2.3, we tuned the OFDM

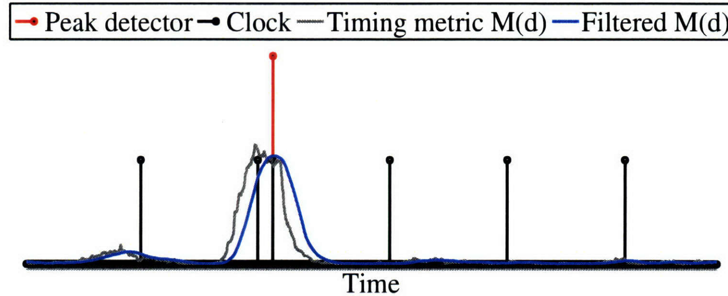


FIGURE 4-2—Correlation between the received OFDM signal and the pseudonoise preamble sequence. A peak in the filtered correlation resets the symbol clock regenerator, as shown here.

synchronization algorithm, in particular improving the peak detection subsystem. The synchronization algorithm accomplishes both symbol timing synchronization and frequency offset compensation, but the former is an easier problem because OFDM symbols are lengthy in time; in contrast, orthogonality between subcarriers may be destroyed by an uncompensated frequency offset.

The Schmidl-Cox algorithm works as follows. The training sequence contains two OFDM symbols, c_1 and c_2. The first, c_1, contains a pseudonoise sequence on the even frequencies, and zeroes on the odd frequencies. By Fourier symmetries, this results in an OFDM symbol that has two identical halves in the time domain [88]. The receiver samples the incoming signal and converts it to a baseband signal y_k, such that each OFDM symbol contains $2L$ samples. Then the synchronizer computes the following timing metric:

$$M(d) = \frac{\left|\sum_{m=0}^{L-1}\left(y^*_{d+m}y_{d+m+L}\right)\right|^2}{\left(\sum_{m=0}^{L-1}|y_{d+m+L}|\right)^2}. \tag{4.1}$$

When aligned with c_1, each term in the numerator of Equation 4.1 will have approximately zero phase (since for complex z, $zz^* = |z|^2$), and so the magnitude of the sum will have a large value. The denominator of Equation 4.1 normalizes $M(d)$ to incoming signal power. The result, shown as the grey curve labeled "Timing metric $M(d)$" in Figure 4-2, is a timing metric that peaks at the beginning of training symbol c_1, which we filter using a moving-average filter to obtain the blue curve labeled "Filtered $M(d)$" in

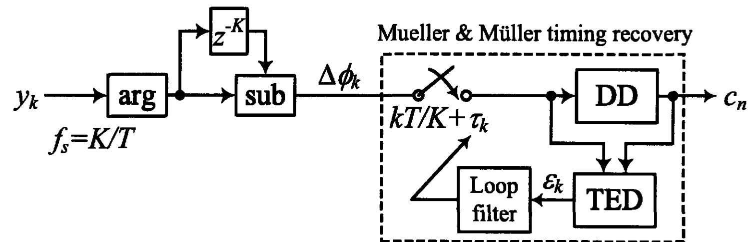


FIGURE 4-3—Block diagram of the feedback-based Zigbee synchronization algorithm. Using chip decisions c_k from the decision device "DD," the timing error detector "TED" produces an error signal ϵ_k which is used to update the sampling phase offset τ_k.

the same figure. The peak detector logic we designed can then pick out the location of c_1, and reset the timing clock, as shown in the figure.

By averaging the phase difference between each sample of the first and second halves of c_1, the receiver can estimate the frequency offset up to an integer multiple of $2/T$, where T is the symbol time. The receiver uses measurements from c_2 to disambiguate this frequency offset estimate; we refer the interested reader to Schmidl and Cox [108, §4] for the details.

4.2.2 Zigbee synchronization

In this section we describe two synchronization designs for Zigbee. The first design is based on a phase-locked loop, and is the design we used to evaluate the system for reliable retransmissions described in Chapter 5.

Feedback loop design

In this Zigbee receiver, we use differential demodulation to detect each MSK chip in a Zigbee codeword (see Section 3.2 for a high-level description of Zigbee spread spectrum coding). Referring to Figure 4-3, complex-valued baseband data y_n comes into the receiver at a sampling rate $f_s = K/T$ where T is the symbol (chip) time.[2] The first stage of the receiver takes the angle of each incoming sample ("arg" box), and differences values in the stream K samples apart, resulting in a stream $\Delta\phi_n$ of angle differences between samples one symbol-time apart.

The key insight to this receiver is a property of the MSK waveform that samples one symbol-time apart have a phase-difference of either $+\pi/2$ or

[2]The Zigbee chip time is $T = 0.5\mu s$.

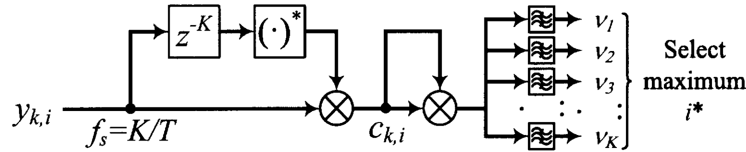


FIGURE 4-4—Block diagram of the Mehlan, Chan, Meyr (MCM) feedforward timing and frequency recovery algorithm as we have implemented it in a software radio platform.

$-\pi/2$ [89]. Thus once we synchronize on the MSK waveform, the output of the receiver, c_k, is a symbol-rate stream of binary data.

To recover the timing of the MSK signal we use the popular Mueller and Müller decision-directed timing recovery algorithm [83, 19], depicted in the dashed box of Figure 4-3. From the incoming data stream $\Delta\phi_k$ with sampling offset τ_k, the decision device "DD" makes chip decisions c_n, which are output at symbol rate $1/T$. The timing error detector "TED" takes the chip decisions and incoming data stream and computes an error signal ϵ_k as follows:

$$\epsilon_k = \Delta\phi_k c_{k-1} - \Delta\phi_{k-1} c_k. \tag{4.2}$$

The TED's error signal is filtered ("Loop filter") and used to update the sampling offset τ_k. Design of the loop filter is a well-studied problem; we refer the interested reader to Gardner's text [38] for more information. Implementation of the sampler component in digital logic is usually handled with a fractionally-spaced digital interpolator; we refer the interested reader to the texts by Meyr et al. [79, Chp. 15] and Mengali [75] for more details.

Feedforward design

In an updated design for synchronization on the Zigbee signal, we use a feedforward timing and frequency recovery algorithm by Mehlan, Chan, and Meyr [74]. The key observation here is two-fold. First, each of the chips in a Zigbee codeword is modulated with minimum-shift keying [89] (see Section 3.2 for further details) and second, when an MSK signal is passed through a fourth-order non-linearity, the resulting signal has certain periodicities that can be exploited for synchronization.

Figure 4-4 shows a high-level block diagram of the algorithm. The incoming baseband signal $y(t)$ is sampled and optionally interpolated to a frequency K times the nominal symbol rate (see Figure 4-1. For the purpose

of acquiring the packet preamble and initial synchronization, the MCM algorithm assumes that T_s is an integer fraction of T.[3] The first structure in Figure 4-4 delays the incoming data stream $y_{k,i} = y\left(\left(k + \frac{i}{N}\right)T\right)$ by a symbol time (K samples), takes the complex conjugate of the delayed data, and multiplies (mixes) it with the original data stream. The result is a second-order (quadratic) stream $c_{k,i}$ whose angle is the difference between successive symbols, at each of K different sampling offsets i. The next structure in the block diagram simply squares each element of this data stream, and the result is demultiplexed and filtered with K independent window-averaging filters, yielding K filtered streams at the right-hand side of Figure 4-4. We then choose the timing offset i^* corresponding to the filter whose output is maximum among the K possibilities. It can be shown [74] that an accurate frequency offset estimate for the incoming packet is

$$\Delta\hat{\omega}T = \frac{\arg\{-\hat{v}_{k,i^*}\}}{2}. \tag{4.3}$$

After selecting symbols at the right timing offsets as described above, the receiver rotates the stream of symbols by $\Delta\hat{\omega}T$ in order to correct for the frequency offset between transmitter and receiver.

We have implemented the MCM algorithm in the USRP software radio [25], and used it to synchronize the USRP to transmissions from the commercially-available Chipcon CC2420 Zigbee radio [117]. Once synchronized, our receiver computes the correlation ρ_k between the frequency-corrected version of the incoming samples c'_{k,i^*} and the differentially-encoded known preamble sequence p_k as follows:[4]

$$\rho_k = \sum_{l=1}^{L} c'_{k,i^*} p_k. \tag{4.4}$$

The result is shown in Figure 4-5. We see distinct peaks in the correlation output, corresponding to those points in time where a preamble or postamble is present in the incoming data stream. As in the OFDM synchronizer discussed above in Section 4.2.1 we can apply a simple peak detector algo-

---

[3]This assumption is valid for the short duration of preambles, and in practice, a PLL would be used to track the symbol clock over the whole duration of the packet, as described above in the previous Zigbee design.

[4]Note that differential detection here is not necessary, but simplifies testing: once the symbol timing and frequency offset corrections have been applied, the receiver could detect with the incoming samples $y_{k,i}$ directly.

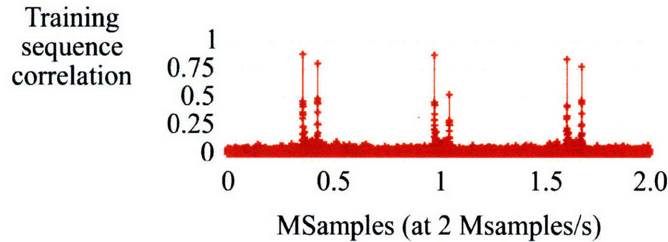


FIGURE 4-5—Correlation between the fixed training sequence and the samples corresponding to the timing indices chosen by the MCM algorithm. Three packets are clearly visible here, with correlation peaks at the beginning and end of each packet corresponding to the preamble and postamble training sequences.

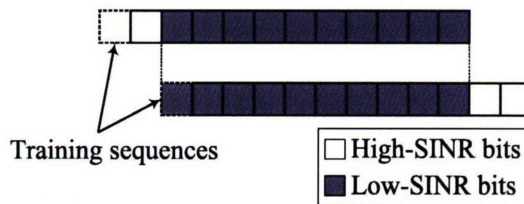


FIGURE 4-6—In status quo systems, training sequences are present only in the preamble of the packet. When packets collide, the training sequences overlap with data from other packets, making them harder to detect.

rithm to the correlation output to find the start and/or end of each frame and start the process of detecting and decoding the data in the frame.

4.3 THE NEED FOR A POSTAMBLE

Having described synchronization in detail for a number of different physical layer implementations, we now motivate the need for synchronization based on a frame postamble. Referring to the diagram of two colliding packets shown in Figure 4-6, notice that the training sequence for the lower packet overlaps in time with the body of the upper packet. This overlap results in a lower SINR for the preamble training sequence of the lower packet.

When the preamble coincides in time with other packets or noise, current radio receivers will not be able to synchronize with the incoming transmission and decode any bits. In that case, the potential benefits of the SoftPHY interface will be lower. We need a way to mitigate the effects of preamble loss in the collision shown in Figure 4-6. In this example, a receiver would

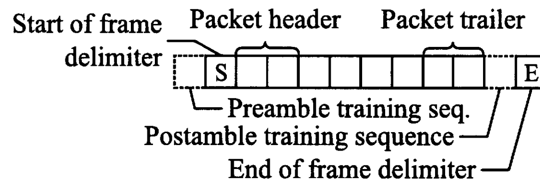


FIGURE 4-7—The packet layout for postamble-based synchronization. Training sequences aid the receiver in detecting the presence of the packet, and start/end of frame delimiters ("S" and "E") assist in framing the packet.

not be able to decode any part of the lower packet, since its preamble was not detected, while the upper packet would be detected and its bits delivered by the SoftPHY interface.

4.4 THE POSTAMBLE-BASED APPROACH

Our approach to synchronizing on packets without an intelligible preamble is to add a *postamble* to the end of each packet on which a receiver can also synchronize. The postamble has a well-known sequence attached to it that uniquely identifies it as the postamble, and differentiates it from a preamble ("E" in Figure 4-7). In addition, we add a *trailer* just before the postamble at the end of the packet, also shown in Figure 4-7. The trailer contains the packet length, source, and destination addresses. Just as with header data, the receiver uses the SoftPHY interface to check the correctness of the trailer.

4.4.1 Postamble-based framing algorithm

We now describe how postamble-based framing and decoding works in our Zigbee implementation (802.11a has a similar structure). Frames whose preambles are detectable (indicated by the presence of a "start-of-frame" marker following the detection of a training sequence) are processed in the usual order, from left-to-right in time. The start-of-frame marker is labeled "S" in Figure 4-8.

To recover the payload after detecting only a postamble (indicated by the presence of an "end-of-frame" marker following a training sequence), the receiver takes the following steps. The receiver continuously maintains a circular buffer of samples of previously-received symbols (even when it has not detected any training sequence). In our implementation, we keep as

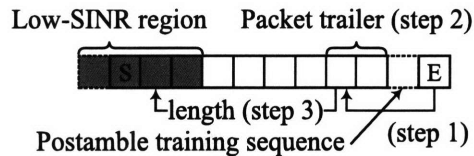


FIGURE 4-8—The postamble-based synchronization algorithm. When the receiver detects an end-of-frame marker ("E" in the figure), it follows the steps outlined in the text to decode as much of the packet as possible.

many sampled symbols as there are in one maximally-sized packet, $K \cdot S$ where K is the number of samples per symbol (typically an integer between two and eight) and S is the packet length in symbols. When the receiver decodes an end-of-frame marker following the detection of a training sequence, it takes the following steps, illustrated in Figure 4-8:

Step 1: "Roll back" in the circular buffer as many symbols as there are in the packet trailer.

Step 2: Decode and parse the trailer to find the starting location of the frame in the circular buffer.

Step 3: "Roll back" in the circular buffer as many symbols as are in the entire packet, and decode as much of the packet as possible.

The main challenge of postamble decoding is addressing how a receiver can keep a modest number of samples of the incoming packet in a circular buffer while still allowing the various receiver subsystems to perform their intended functions. These functions include carrier recovery, symbol timing recovery, and equalization. We meet each of these challenges in our three implementations, as we now explain.

Postamble-based synchronization in the 802.11a OFDM receiver In the design of Section 4.2.1, we maintain a circular buffer of $S \cdot K$ samples as described above, and upon encountering a preamble, we run the OFDM equalizer backward in the sample buffer as described in the next section.

Postamble-based synchronization in the Zigbee feedback loop receiver. In the feedback-loop design of Section 4.2.2, the Mueller and Müller algorithm does not rely on the presence of a preamble, so we implement our circular buffer with storage for exactly S binary symbols, which the receiver examines upon encountering a postamble "end-of-frame" marker.

Postamble-based synchronization in the Zigbee feedforward receiver. The feedforward design of Section 4.2.2 will identify both preambles and postambles, and the receiver may take a similar approach to the 802.11a receiver, as described in the next section.

Complications from postamble detection Techniques for countering inter-symbol interference (equalization) and compensating for a frequency and phase offset often rely on estimating the channel impulse response [6]. Typically the preamble includes a known *training sequence* to enable the equalizer to quickly estimate the channel's response during synchronization. The receiver updates this estimate as it processes the packet from left-to-right in time. When we include the same training sequence in the postamble (see Figure 4-8) we can amend step (3) of the postamble decoding algorithm to post-process the samples of the body of the packet backward in time, instead of forward in time. This step allows the phase-locked loop in a feedback-based equalizer to track the channel impulse response starting from a known quantity estimated from the postamble.

4.5 CODEWORD-LEVEL SYNCHRONIZATION

In this section we integrate the concepts from the Zigbee SoftPHY implementation (Section 3.2.2, page 57), Zigbee symbol synchronization (Section 4.2.2), and postamble-based synchronization (Section 4.4). Recall from Section 3.2.2 that each Zigbee codeword is composed of 32 MSK symbols. To correctly decode a Zigbee codeword, the receiver must synchronize on both the codewords as they are aligned in the stream of symbols, and the symbols themselves. We accomplish this by testing all possible codeword synchronization *offsets* for a preamble or postamble, and then decoding only at the particular offset at which the preamble or postamble was detected. The following discussion, however, shows that the postamble gives the codeword synchronization process more resilience.

Figure 4-9 shows a receiver's view of a single packet at two different codeword synchronization offsets. The packet contains a known bit pattern, against which we test each received codeword for correctness. The result of each test is indicated by the presence or absence of the labeled box in the figure.[5] The upper plot in Figure 4-9 shows the packet arriving at the receiver at time[6] 0, and achieving synchronization at time 10 (lower plot). When

[5] For clarity, we show the result of every fourth codeword-correctness test.

[6] Measured in units of codeword-time, 16 μs in our radios.

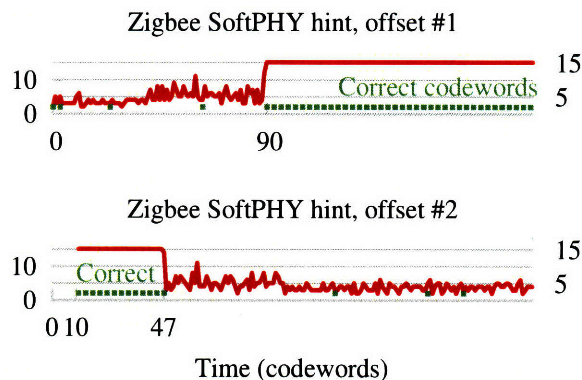


FIGURE 4-9—Partial packet reception of a single packet, at two different codeword synchronization offsets during a loss in codeword-level synchronization. We show codeword correctness (box indicators) and each codeword's associated SoftPHY hint (curves). Despite uncertainty in physical layer codeword synchronization, the SoftPHY hint indicates the correct parts of the packet to higher layers.

the PHY synchronizes on the packet, symbol timing recovery succeeds and the receiver decodes approximately 40 codewords correctly (including the preamble) before losing symbol or codeword synchronization. We see that the SoftPHY hint remains at 15 for the duration of the correct codeword decisions, and falls at time 47 when the burst of errors occurs. As described in Chapter 3, the physical layer passes these SoftPHY hints up along with all the data bits in the packet.

Later, at time 90 at the other synchronization offset (upper plot), the receiver successfully synchronizes on and decodes a run of codewords extending to the end of the first packet. Since this packet data is at a different synchronization offset to the preamble, it relies on its postamble in order to frame-synchronize and pass up the partial packet reception and associated SoftPHY hints.

4.6 RELATED WORK

The present work is the first we are aware of that integrates postamble decoding in the context of a larger system for recovering parts of packets, as part of a large multihop wireless system. Detection is a well-studied area, however, and there is a wealth of work relating to the basic concepts we describe in this chapter.

Code-aided turbo synchronization [46] and iterative timing recovery [7] take the structure of coding into account in the synchronization process. There have been a number of synchronizer designs that use these principles [84]. In an iterative or loop design, these techniques estimate the timing of the incoming symbols, and then make a soft decoding attempt at the data using a soft-input, soft-output decoding algorithm like BCJR [5]. Then these techniques use the soft outputs of the decoding algorithm to make a better estimate of symbol timing, and use the updated timing estimates to make a more accurate decoding attempt. The process continues in an iterative fashion. These techniques are complementary to the use of a postamble for packet detection because they rely a priori on detecting the presence of a packet. They could be used in conjunction with postamble packet detection for even better performance.

Also in the context of turbo decoding algorithms, Godtmann et al. [40] independently propose several training sequence positioning schemes, including preamble plus postamble, preamble-only, "mid-amble," and distributing the training symbols throughout the body of the packet. The authors investigate these schemes in the context of a turbo decoder, a receiver design for decoding Turbo-coded data [8].

Gansman et al. [33] show that distributing the training sequence at the beginning and end of the packet is optimal for the purpose of performing the most accurate frequency offset correction possible. This result makes intuitive sense because the amount of frequency drift between any two points in the packet attains its maximum at the beginning and end of the packet burst transmission. In the same vein and with slightly more generality in their results, Noels et al. [85] investigate the influence of the location of pilot symbols on the Cramér Rao lower bound on the variance of a joint estimate of carrier phase and frequency offset.

Whitehouse et al. [127] and independently, Priyantha [96] propose techniques for avoiding "undesirable capture" in wireless networks. Undesirable capture occurs when the weaker of two packets arrives first at a receiving node, so that the receiver attempts to decode the earlier and weaker packet and not the stronger and later packet, which corrupts the decoding attempt, resulting in neither being decoded correctly. With the postamble, the receiver makes a decoding attempt on both packets.

Finally, beyond synchronization, direct sequence spread-spectrum radios face an additional problem of spreading code acquisition: aligning the incoming data with the pseudorandom noise spreading sequence [95]. Since the entire body of a direct sequence spread spectrum data packet is modu-

lated using a known spreading sequence, Jeong and Lehnart propose using all of the packet for the related problem of spreading code acquisition [60].

5

Reliable Retransmissions with SoftPHY

THE MECHANISMS we have thus far proposed, SoftPHY in Chapter 3 and postamble detection in Chapter 4, work together in the following three ways. They allow the receiver to (*i.*) detect more transmissions, (*ii.*) recover more bits from the transmissions which are detected, and (*iii.*) determine which of the received bits are likely to be correct and which are likely to be incorrect. This and the next chapter focus on concrete uses for these mechanisms in wireless networking protocols.

In this chapter, we examine the problem of how a sender and receiver can use the above mechanisms to improve the throughput of an automatic repeat request (ARQ) protocol that accomplishes link level reliable retransmissions. The protocol, *partial packet ARQ* (*PP-ARQ*) is a variant of ARQ that uses *partial packets*, our term for the fragments of packets that the SoftPHY interface passes up to the link layer.

Chapter overview. We begin with a high-level picture of the PP-ARQ protocol, outlining how a sender and receiver can work together to achieve link-layer reliability. We then formulate the problem of how the receiver can best retransmit partial packets as a dynamic programming problem in Section 5.2. Next, we note the drawbacks of PP-ARQ thus far described as a result of feedback channel utilization and header overheads, and propose our final protocol, *PP-ARQ with packet streaming* as a solution in Section 5.3. We then describe our implementation and present our evaluation in Sections 5.4 and 5.5, respectively. The chapter concludes with a look at work related to PP-ARQ.

5.1 THE PP-ARQ PROTOCOL

In Chapter 3 we introduced the SoftPHY interface, which passes up confidences of each bit's correctness to higher layers. However, in Section 3.3, we saw that sometimes SoftPHY hints are incorrect under the threshold test for labeling bits "bad" or "good." Partial packet ARQ (PP-ARQ) is a protocol that builds on a SoftPHY-enabled physical layer, using checksums in a novel way to ensure that the data passed up to the network layer is correct, to the extent that the underlying checksum provides this confidence.

A naïve way to approach the problem is for the receiver to send back the bit ranges of each part of the packet believed to be incorrect. Unfortunately, doing that consumes a large number of bits, because encoding the start of a range and its length can take on the order of $\log S$ bits for a packet of size S. Furthermore, as we note in Section 5.5, most error burst events in our Zigbee experiments are small, 1/16th the size of a packet. Hence, we seek a more efficient algorithm that reduces feedback overhead.

At a high level, we summarize the steps in the PP-ARQ protocol as follows:

Step 1: The sender transmits the full packet with checksum.

Step 2: The receiver decodes the packet or part of the packet using SoftPHY and postamble detection as described in Chapters 3 and 4, respectively.

Step 3: The receiver computes the best feedback as described in Section 5.2.

Step 4: The receiver encodes the feedback set in its reverse-link acknowledgment packet (which may be empty, if the receiver can verify the forward link packet's checksum).

Step 5: The sender retransmits only (a) the contents of the runs the receiver requests, and (b) checksums of the remaining runs.

Step 6: The receiver combines the transmissions received thus far from the sender.

This process continues, with multiple forward-link data packets and reverse-link feedback packets being concatenated together in each transmission, to save per-packet overhead.

Protocol overview. To motivate our discussion, we consider the example in Figure 5-1. The sender sends its first transmission of the entire data

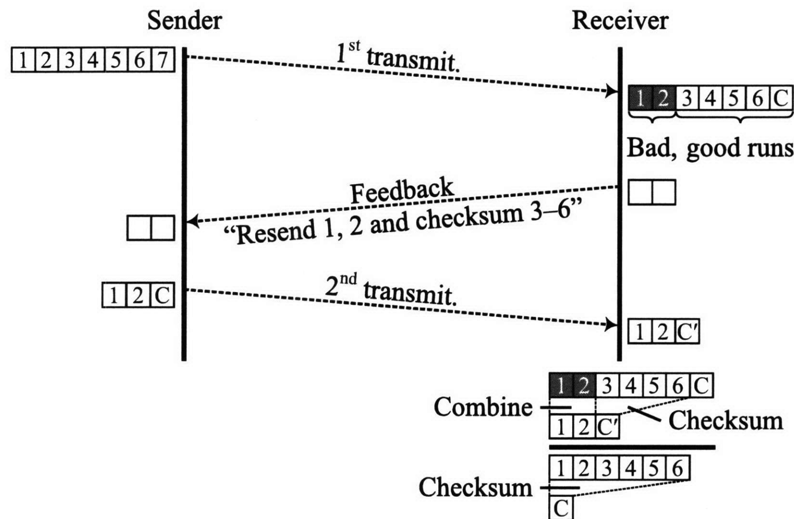


FIGURE 5-1—An exemplary run of the PP-ARQ protocol, showing the first packet transmission, receiver-side labeling of the packet using SoftPHY and the threshold test, synchronous feedback, and the second packet transmission containing data bits and partial-packet checksums.

packet, blocks 1–7 in the picture. Note that these data blocks are sent and received using a status quo physical layers, that apply some amount of coding redundancy to the transmissions, but inevitably bits get corrupted, as discussed in Chapter 1.

Upon receiving the first transmission, the receiver makes a packet decoding attempt, labeling each block in the packet as either "bad" or "good" using the SoftPHY interface and the threshold test. Then, it builds a *feedback* packet based on its labeling. The feedback packet asks for all of the "bad" bits to be retransmitted, and asks for checksums over all of the "good" bits. Once the receiver has computed its feedback packet, it replies to the sender synchronously (within a fixed time bound), at which point the sender transmits all of the "bad" bits requested (blocks one and two), and a checksum of the "good" bits (C') in its second transmission.

Upon receiving the second transmission, the receiver verifies the checksum C' over blocks 3–6, and combines the two transmissions of blocks one and two in some manner. In our implementation we use a simple replacement policy for combining multiple transmissions of the same block, but more sophisticated techniques are available, which we discuss in Section 5.6

below. Finally, once the receiver has assembled the two transmissions, it verifies the packet checksum C over the entire packet (1–6).

This high-level overview of the protocol brings to attention several other questions. First, how does the receiver avoid asking for every single run of "bad" bits in the packet when "bad" bits are finely interleaved with "good" bits? Second, how does the receiver detect and recover from situations in which bits labeled "bad" are in fact correct, or bits labeled "good" are in fact incorrect? And finally, what happens when either of the checksums C or C' fails, and with what frequency do either of those events occur? We answer these questions in the following discussion, backing up our responses with empirical evidence.

5.1.1 PP-ARQ at the receiver

Since the salient feature of bit error patterns is the length of the various "good" and "bad" runs within a packet, we now introduce notation that reflects this. Recall from Chapter 3 that once the physical layer has decoded a packet of length L, the receiver has a list of received symbols[1] $\mathbf{S}_i$, $1 \leq i \leq L$, and SoftPHY hints φ_i. After using the threshold test to label each symbol "good" or "bad, it computes alternating *run lengths* $\mathbf{\Lambda^g} = \lambda_j^g$, $\mathbf{\Lambda^b} = \lambda_j^b$, $1 \leq j \leq R$ of "good" and "bad" symbols, respectively, where a *run* is defined as a number of symbols sharing one label ("good" or "bad"). We denote the number of runs in the packet R.

The run-length representation. The receiver thus forms the *run-length representation* of the packet as shown in Figure 5-2. This representation has the form

$$\lambda_1^b \lambda_1^g \lambda_2^b \lambda_2^g \cdots \lambda_R^b \lambda_R^g. \tag{5.1}$$

Here, λ_j^g is the list of symbols in the jth run of symbols all labeled "good," shown with light shading in the figure. Similarly, λ_j^b is the list of the jth run of symbols labeled "bad," shown with dark shading in the figure. Note the ordering of "bad" and "good" runs, starting with a "bad" run, which may be of length zero in the event that the packet starts with a "good" symbol.

Which bits to ask for? We call the groups of bits which the receiver does and does not request retransmission of *bad chunks* and *good chunks*, respectively. Clearly the union of all the bad chunks must contain all the bad

[1] Here we use the term symbol to mean the unit of granularity with which the physical layer makes bit decisions, b in Chapter 3.

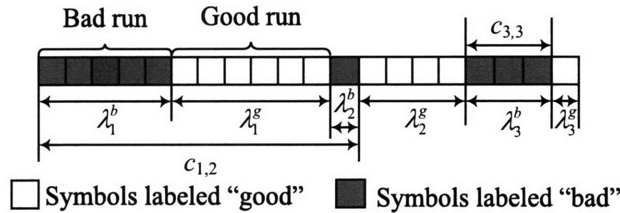


FIGURE 5-2—The run length representation of a received packet, and associated notation.

runs, but are there any other constraints on which bits to ask for? We now show that assuming the correctness of SoftPHY hints, optimally-chosen bad chunks must start on the boundary of a transition from "good" to "bad" bits, and end on the boundary of a transition from "bad" and "good" bits.

> *Proof.* We make the argument in two steps: first, we show that a bad chunk may not begin nor end with "good" bits. Next we show that bits immediately preceding and following a bad chunk must be "good." The two steps together prove the result.
>
> To show the first step, consider a bad chunk that begins or ends with a l "good" bits. Assuming that the SoftPHY labels at the receiver are correct, we could reduce the size of the consequent forward-link retransmission by truncating the l "good" bits from either the beginning or end.
>
> To show the second step, first notice that the bad chunks the receiver asks for have to contain every "bad" run in the received packet. Now consider a bad chunk with l "bad" bits immediately preceding or following it. If this is the case, we can form a larger bad chunk by concatenating the l "bad" bits to it. Note that the original chunking required the receiver to make two requests for the same bits, one for the original and one containing the l "bad" bits. The resulting chunking choice will therefore be more efficient, since the receiver only has to make a single request. □

The feedback chunking. The receiver then forms a list C of chunks, groups of runs numbering $C \leq R$ which the receiver will ask the sender to retransmit. Chunk c_{i_k,j_k} contains all the "bad" and "good" runs in between and including "bad" run i_k and "bad" run j_k, so each chunk starts and ends

with "bad" runs. More precisely, for $k \in [1, C]$:

$$C = \left\{c_{i_k, j_k}\right\} = \left\{\lambda^b_{i_k} \lambda^g_{i_k} \lambda^b_{i_k+1} \lambda^g_{i_k+1} \cdots \lambda^b_{j_k}\right\} \tag{5.2}$$

For example, chunks $c_{1,2}$ and $c_{3,3}$ appear in Figure 5-2. Note that chunk $c_{i,j}$ does not include $\lambda^g_{j_k}$, the last run of "good" symbols in the chunk. This run of "good" symbols is the run whose checksum the sender will transmit, so that its correctness may be validated.

Balancing feedback resolution with forward-link wastage. Note that the requests the receiver makes need not just contain every "bad" run in the packet. In general, the receiver might request that some good runs be retransmitted as well, for the following reason. Suppose a number of small "good" runs were interleaved with longer "bad" runs (λ^g_k small and λ^b_k large for $i \le k \le j$). In this situation the amount of information needed to describe each of the "bad" runs' offsets may exceed the number of "good" bits contained in the chunk $c_{i,j}$ that would be transmitted a second time. Consequently the receiver should request that the entire chunk be retransmitted. On the other hand, suppose the "good" runs are long relative to the bad runs (λ^g_k large and λ^b_k small for $i \le k \le j$). In this situation we would instead favor asking for the individual chunks $c_{k,k}$ for each $k \in [i, j]$ for the converse reason: the amount of information that would be retransmitted if we did not would be excessive.

The receiver thus has two choices for each chunk $c_{i,j}$ in the packet:

1. *Merge* the chunk, asking the sender to retransmit all "good" bits contained within it.

2. *Split* the chunk at some chunk index k, asking the sender to retransmit only the "bad" bits in $c_{i,k}$ and $c_{k+1,j}$ respectively.

Feedback data layout. Once the receiver has made the choice of which chunks to request from the sender, it sends the *feedback data fragment* shown in Figure 5-3 communicating this information to the sender. The feedback data fragment contains packet and transmission sequence numbers so that the sender can identify the forward-link packet and transmission that the feedback data fragment refers to. This is necessary because senders and receivers may group feedback data fragments belonging to more than one forward-link packet in the same transmission for the reasons described below in Section 5.3. The feedback data fragment also contains all chunks' (C

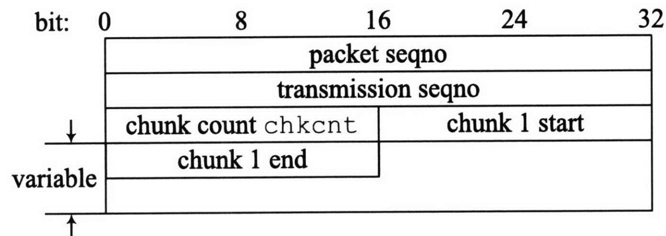


FIGURE 5-3—Byte layout of the packet feedback element that the receiver transmits to the sender.

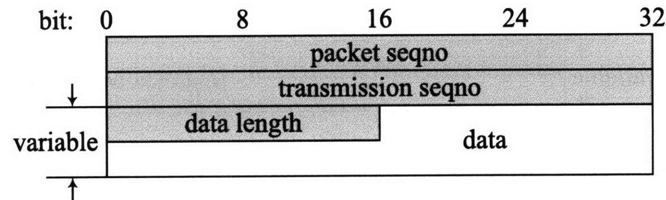


FIGURE 5-4—Byte layout of the forward-link data transmission. Shaded fields in this and subsequent packet layout figures indicate overheads unique to PP-ARQ, which we measure in Section 5.5.

in Equation 5.2) beginning and ending symbol offsets, determined by the dynamic programming algorithm discussed in the next section.

The receiver sends the feedback packet synchronously, in a prioritized timeslot immediately after the forward-link data transmission, in the same manner as IEEE 802.11 [50]. In the performance evaluation below we evaluate the feedback data overhead to PP-ARQ.

5.1.2 PP-ARQ at the sender

In its first transmission to the receiver, the sender transmits simply the data, as shown in Figure 5-4. The first transmission of a given packet consists of the packet sequence number, transmission sequence number (always set to zero), the data length, followed by the data itself.

The sender buffers the current packet in memory, so that when it receives a feedback packet, it can construct subsequent retransmissions, which we now describe. From the feedback packet it receives, the sender reads each of the chunks' starting and ending offsets. Once the receiver has these offsets, it builds a *forward-link data retransmission* as shown in Figure 5-5. The forward-link retransmission begins with the packet and transmission

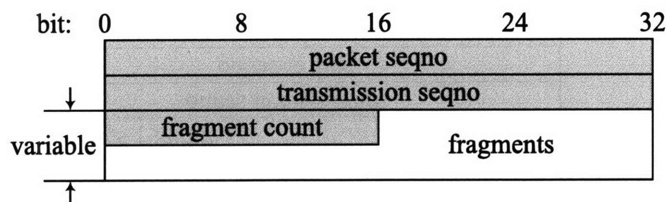


FIGURE 5-5—Byte layout of the forward-link data retransmission. The `fragments` field contains multiple forward-link data retransmission fragments whose structure is shown in Figure 5-6 below.

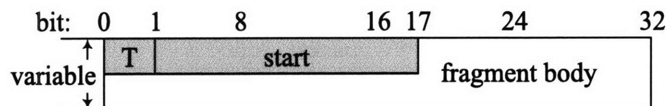


FIGURE 5-6—Byte layout of the forward-link data retransmission fragment. The `fragment body` field contains one of a good or bad fragment body, the former Figure 5-7 illustrates, and the later the text specifies.

sequence numbers `packet seqno` and `transmission seqno` respectively, needed for the receiver to determine the data packet the retransmission corresponds to. The next field in the retransmission is a 16-bit count of the number of fragments contained in the retransmission, which assists the receiver in parsing the fragments which follow. This field is followed by the fragments themselves, arranged contiguously in the `fragments` field.

Each *retransmission fragment* (see Figure 5-6) contains a header with a one-bit field `T`, representing the type of fragment, following by a 16-bit `start` field indicating the bit offset within the packet at which the fragment begins. All fragments in a given retransmission belong to the same packet. The type of the fragment determines the contents of the `fragment body` field which immediately follows the `start` field. Fragments can be one of two types:

1. *Good fragment* (`T = 0`). The `fragment body` consists of a 32-bit checksum taken over the (known correct) packet data at the sender, beginning at the starting offset specified in the `start` field of the fragment header in Figure 5-6 and ending at but not including the starting offset of the succeeding fragment header, or the end of the packet if the fragment is the last in the packet.

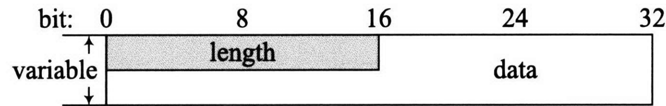


FIGURE 5-7—Byte layout of forward-link retransmission bad fragment body (T = 1 in Figure 5-6). The good fragment body contents are described in the text.

2. *Bad fragment* (T = 1). The `fragment body`, illustrated in Figure 5-7, begins with a 16-bit field `length` which indicates the length in bits of the following `data` field. The `data` field contains retransmitted data from the original packet, starting at the bit offset `start` in the containing fragment header, and `length` bits in length.

The dark fields in Figures 5-4, 5-5, 5-6, and 5-7 are metadata needed for PP-ARQ's internal bookkeeping. We measure their performance overhead below in Section 5.5.

★ ★ ★

Figure 5-8 shows an example packet reception, with each possible chunk labeled. We immediately see that many chunks are contained within larger chunks, for example chunk $c_{1,3}$ contains chunk $c_{1,2}$. Furthermore, the optimal solution for a chunk contains the optimal solutions for all the chunks contained within in (otherwise we could improve the s). We also immediately see that many chunks overlap with each other ($c_{1,2}$ and $c_{2,3}$ for example). These two properties, overlapping subproblems and optimal substructure, are the hallmarks of dynamic programming, the subject of the next section.

5.2 TUNING FEEDBACK WITH DYNAMIC PROGRAMMING

As we noted in Section 5.1.1, the receiver has a choice of which chunking to choose. We now show how to assign each chunking a cost function that indicates the amount of additional bit-overhead that the chunking incurs. Once we formulate a cost function, we can apply standard dynamic programming (DP) techniques [17] to minimize the cost function and associated overhead.

A cost function for DP. We define the *chunk cost function* **C** of a chunk $c_{i,j}$ in two steps. First, the base case in which $i = j$ indicates a chunk that

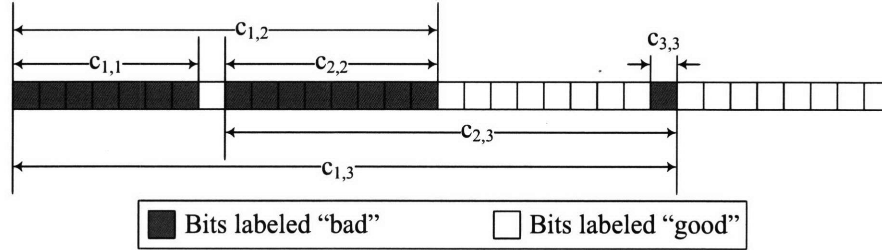


FIGURE 5-8—A motivating example for the PP-ARQ dynamic programming algorithm. We annotate every possible chunk that the PP-ARQ DP algorithm considers in this packet.

contains a single "bad" run of bits, such as $c_{1,1}$, $c_{2,2}$, or $c_{3,3}$ in Figure 5-8. Then we consider larger chunks in the recursive case.

In the base case, the receiver must describe the length and offset of the ith bad run to the sender. This takes approximately $2 \log L$ bits on the forward channel, where L is the packet length defined on p. 90. The receiver also sends a checksum of the ith "good" run to the sender, so that the sender can verify that it received the good run correctly. This takes $\min\left(\lambda_i^g, \kappa\right)$ bits on the feedback channel, where κ is the length of the checksum. The cost function for the base case chunk is therefore

$$C\left(c_{i,i}\right) = 2 \cdot \log L + \min\left(\lambda_i^g, \kappa\right). \tag{5.3}$$

In the recursive case, the receiver must choose one of two options: either *split* the chunk and request two of its sub-chunks individually, or *merge* the chunk, incurring the penalty of retransmitting all "good" bits contained within. In the former case, the receiver must also choose at which run l the chunk is to be split.

In the case that the receiver chooses to merge the chunk, it incurs an additional bit-overhead equaling $2 \log L$ to describe the chunk offsets, plus the number of good bits contained within the chunk. It also incurs the overhead needed to send a checksum of the jth "good" run following the bad chunk:

$$C_{merge}\left(c_{i,j}\right) = 2 \log L + \sum_{l=i}^{j-1} \lambda_l^g + \min\left(\lambda_i^g, \kappa\right). \tag{5.4}$$

In the case that the receiver chooses to split the chunk into two smaller chunks, the cost is the sum of the costs of the two smaller chunks, minimized over all possible choices of where to make the split, if $j - i \geq 2$. The cost in

the splitting recursive case is therefore

$$C_{split}\left(c_{i,j}\right) = \min_{i \le k \le j-1} \left\{ C\left(c_{i,k}\right) + C\left(c_{k+1,j}\right) \right\}. \tag{5.5}$$

Note that the costs of checksumming the kth and jth "good" runs are included in the individual costs, so we do not include them in Equation 5.5.

The final cost in the recursive case for chunk $c_{i,j}$ is the minimum of the costs of each alternative:

$$C\left(c_{i,j}\right) = \min \left\{ C_{merge}\left(c_{i,j}\right), C_{split}\left(c_{i,j}\right) \right\}. \tag{5.6}$$

A bottom-up DP algorithm. Having defined the cost function, we present in Figure 5-9 a dynamic programming algorithm that works "bottom-up," first from the base case at line 1 of chunks consisting of a single bad run, and then in the recursive case, starting at line 4. BOTTOM-UP-PP-ARQ-DP takes as input good and bad run lengths $\mathbf{\Lambda^g}$ and $\mathbf{\Lambda^b}$, respectively, the number of runs R, and the checksum size κ. The recursive case code starting on line 4 first computes the cost function in the splitting case (Equation 5.5) with the **for** loop at line 9. It stores the result in *split-cost* and the split point that generated that result in *arg-split-cost*. Then at line 13 it evaluates the cost of merging the chunk instead (Equation 5.4), and chooses the minimum of the two alternatives, implementing the minimization in Equation 5.6. It then stores the results into the $\mathbf{C}$ and $\mathbf{\Pi}$ matrices for the chunk being evaluated. BOTTOM-UP-PP-ARQ-DP terminates when it completely fills in two matrices: a *cost* matrix $\mathbf{C}$ that gives the costs of every possible chunking, and a predecessor matrix Π that gives the chunk splits or merges which yield the optimum cost of a particular chunking.

We follow the execution of BOTTOM-UP-PP-ARQ-DP as it processes the packet in Figure 5-8 on p. 96, showing the resulting cost and predecessor tables in Figure 5-10. BOTTOM-UP-PP-ARQ-DP first fills in the bottom elements of the cost and predecessor tables of Figure 5-10; to simplify our exposition we normalize these costs to unity. Since chunks containing only a single "bad" run should not be split (as argued in §5.1.1) we set base-case chunks' predecessor entries to *undef* accordingly. In the recursive case, BOTTOM-UP-PP-ARQ-DP first examines chunk $c_{1,2}$ ($i = 1$, $j = 2$ in Figure 5-10), choosing to merge it, and setting $\Pi_{1,2}$ *undef* accordingly. When it examines chunk $c_{2,3}$ next it makes the opposite choice, splitting the chunk and recording the split point, 2, in $\Pi_{2,3}$. This stands in agreement with intuition, since there is a long run of "good" bits at that point in Figure 5-8. In the final recursive step, BOTTOM-UP-PP-ARQ-DP examines chunk $c_{1,3}$

```
BOTTOM-UP-PP-ARQ-DP(Λ^b, Λ^g, R, κ)
 1  for i ∈ [1, R] ▷ Base case
 2      do C_{i,i} ← C(c_{i,i})
 3         Π_{i,i} ← undef
 4  for l ∈ [1, R − 1] ▷ Recursive case: for increasing chunk lengths
 5      do for i ∈ [1, R] ▷ Loop over start point
 6             do j ← i + l
 7                split-cost ← ∞
 8                arg-split-cost ← 0
 9                for k ∈ [i, j − 1] ▷ Loop over split point
10                    do if C_{i,k} + C_{k+1,j} < split-cost
11                          then split-cost ← C_{i,k} + C_{k+1,j}
12                               arg-split-cost ← k
13                merge-cost ← C_merge(c_{i,j})
14                if split-cost < merge-cost
15                  then C_{i,j} ← split-cost
16                       Π_{i,j} ← arg-split-cost
17                  else C_{i,j} ← merge-cost
18                       Π_{i,j} ← undef
19  return (C, Π)
```

FIGURE 5-9—Pseudocode for a dynamic programming implementation of PP-ARQ. BOTTOM-UP-PP-ARQ-DP returns a predecessor matrix Π that tells the receiver which chunks to split and which to merge in order to form the receiver feedback packet as described in Section 5.1.1.

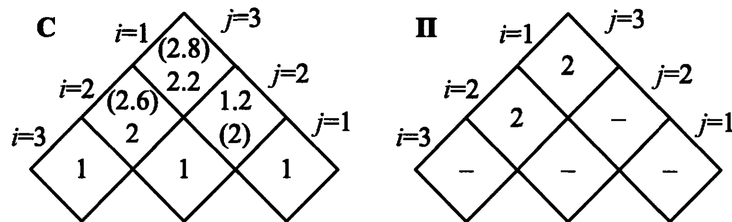


FIGURE 5-10—PP-ARQ dynamic programming receiver-side feedback computation cost **C** and predecessor **Π** matrices for the example packet reception shown in Figure 5-8. In each cost matrix cell, C_{merge} is shown above C_{split}, and the choice BOTTOM-UP-PP-ARQ-DP avoids is shown in parentheses. Base-case costs are shown normalized to unity.

```
GEN-FEEDBACK (Π, i, j)
1  if Π_{i,j} = undef
2     then return {(i, j)}
3     else k ← Π_{i,j}
4          return GEN-FEEDBACK (Π, i, k) ∘
                  GEN-FEEDBACK (Π, k + 1, j)
```

FIGURE 5-11—Pseudocode to generate a list of chunks for inclusion in the feedback message from the predecessor matrix produced by BOTTOM-UP-PP-ARQ-DP.

(which corresponds to the entire packet), and it makes the same choice as before, to split the chunk at 2.

Computational complexity. Note that because BOTTOM-UP-PP-ARQ-DP operates on chunks, the cost and predecessor tables **C** and **Π** have linear dimensions equal to the number of chunks in the packet, R. The base case of BOTTOM-UP-PP-ARQ-DP, starting on line 1, loops through the run lengths once, in time proportional to R. The recursive case, starting on line 4, nests three loops at lines 4, 5, and 9, each of which contains up to R steps, for a total complexity cubic in R. In the evaluation section below, we empirically measure R.

Constructing the feedback message from Π. To form its feedback message, the receiver first runs BOTTOM-UP-PP-ARQ-DP to generate the predecessor matrix **Π**. Then the receiver runs GENERATE-FEEDBACK-CHUNKING($\Pi, 1, R$) to generate a list of chunks for inclusion in the feedback message, as shown in Figure 5-3. GENERATE-FEEDBACK-CHUNKING is shown in Figure 5-11; it is a simple recursive function that builds a list of feedback chunks. If the predecessor matrix is undefined, GENERATE-FEEDBACK-CHUNKING returns a pair of indices corresponding to the merged chunk. Otherwise, GENERATE-FEEDBACK-CHUNKING recursively calls itself with the two chunks indicated by the split point defined in $\mathbf{\Pi}_{i,j}$, returning the concatenation of the two lists.

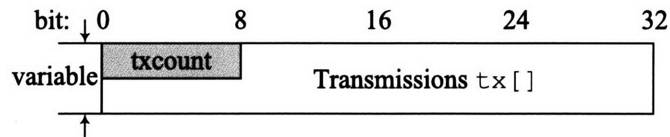


FIGURE 5-12—Byte layout of the streaming PP-ARQ forward-link packet: each of the `txcount` constituent `transmission` fields follows the format in either Figure 5-4 (for the first transmission of a packet) or Figure 5-5 (for subsequent retransmissions).

5.3 ASYNCHRONOUS PACKET STREAMING

As we will show in the evaluation, PP-ARQ as described above does a good job of reducing the size of retransmissions when bit errors occur. The drawback, however, is that successive retransmissions become increasingly smaller as the receiver gains more confidence in more of the constituent bits of the packet, sets SoftPHY hints higher, and consequently labels more bits "good." As retransmissions become smaller, the fixed preamble, postamble, and header overhead associated with each transmission increases relative to the size of the data payload, decreasing throughput. We now present techniques to address this problem.

Motivated by an increasing ratio of header length to payload length, our goal is to combine data from different packets into one transmission, amortizing the header overhead over the longer combined transmission. We therefore use the sliding window algorithm [93] to simultaneously manage the state associated with several packets, concatenating the partial retransmissions together on each successive transmission. Figure 5-12 shows the resulting frame structure of each streaming PP-ARQ forward-link transmission. The frame consists of an eight-bit count `txcount` of the number of constituent transmissions (denoted `tx[]`). Each constituent transmission is either an initial data transmission as shown in Figure 5-4, or a retransmission consisting of multiple fragments (shown in Figure 5-5).

Reverse-link feedback is handled in a similar manner, with `fbcount` feedback fields `fb[]` contained in a single feedback transmission, as shown in Figure 5-13. Each constituent feedback field (denoted `fb[]` in the figure) is a list of chunks, as shown in Figure 5-3.

Figure 5-15 shows an example of how PP-ARQ with streaming amortizes header and preamble overhead. The sender by sending the first packet to the receiver, which labels and packet and sends feedback to the sender asking for a small "bad" run to be retransmitted. In its second transmission,

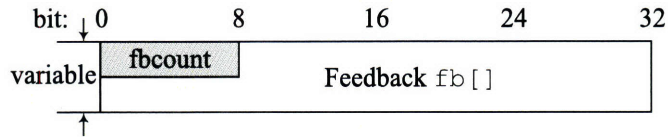


FIGURE 5-13—Byte layout of the streaming PP-ARQ reverse-link packet. Each of the `fbcount` constituent `feedback` fields follows the format shown in Figure 5-3.

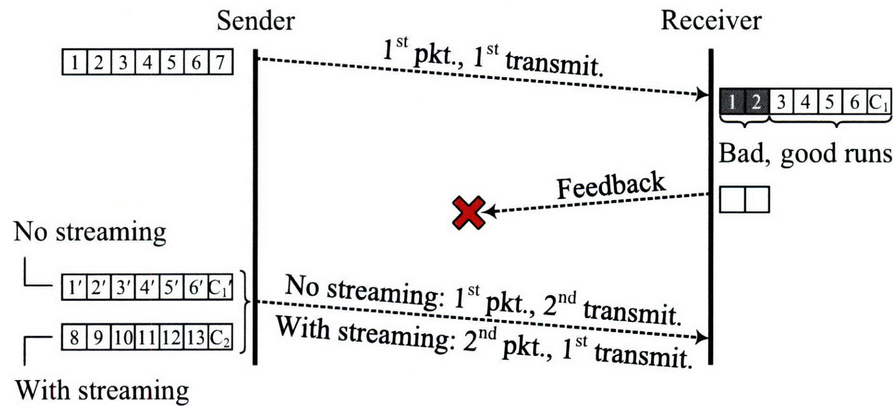


FIGURE 5-14—An example illustrating the operation of PP-ARQ versus PP-ARQ with streaming in the presence of feedback channel loss.

the sender does exactly that, as well as concatenating onto that transmission the data from the second packet.

Streaming acknowledgments also mitigate loss on the reverse channel, as follows. Suppose a feedback packet is dropped as shown in Figure5-14. In that case, without streaming acknowledgments, the sender would timeout and retransmit the frame, losing the benefits of PP-ARQ. With streaming acknowledgments enabled, the sender can continue transmitting packets as shown in the figure, and when the receiver notices that the sender has not received the feedback for packet 1, it can retransmit the feedback, at which point the sender and receiver can continue progress on the first packet.

Sender-side protocol. To implement the sliding window algorithm, the sender maintains two variables: *LPS*, the sequence number of the last packet sent, and *LPA*, the sequence number of the last packet whose contents were fully acknowledged. The sender also maintains a semaphore variable, *send_semaphore* that starts initialized to *WINDOW_SIZE*, the maximum

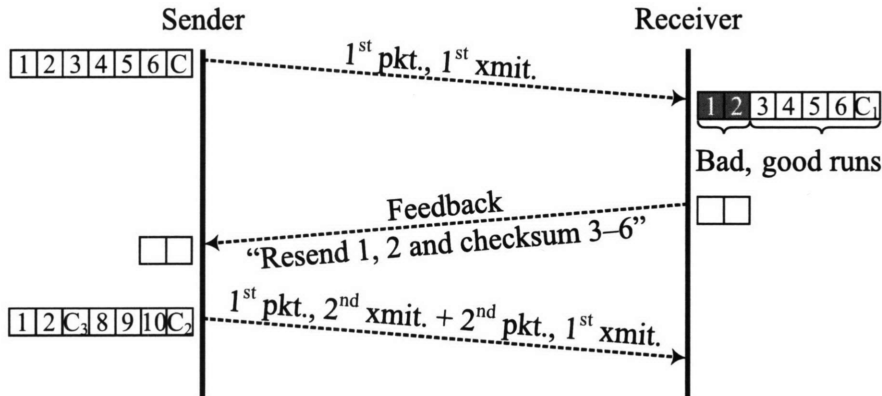


FIGURE 5-15—An example illustrating the operation of PP-ARQ with streaming. In its second transmission, the sender combines a fragment of data from the first packet (blocks 1 and 2) with the second packet's data (blocks 8–10), reducing header overhead.

window size for the sliding window algorithm, and maintains the invariant that $0 \leq LPS - LPA \leq WINDOW_SIZE$. The variable *sendq* manages the run-length representations (Equation 5.1) of multiple packets at the sender.

Figure 5-16 shows pseudocode for the forward-link functionality at the sender. Procedure PP-ARQ-SEND is run when the network layer hands a packet to PP-ARQ at the link layer to send. PP-ARQ-SEND initializes a buffer entry *bufent* with the packet's data, packet sequence number, and packet transmission count, and marks it as ready to send (lines 2–6). Then in line 7 it enqueues the new packet onto the sender's buffer *sendbuf* and resumes the transmit queue if it is not already running.

When RESUME-XMIT-QUEUE executes, we walk through the send buffer *sendbuf* and build a transmission *tx* out of fragments of all the packets waiting in the sender buffer *sendbuf*. We start at line 1 by initializing the number of constituent data packets in the transmission to zero. Then we loop through the send buffer (line 2) looking for packets whose ready to send field *rts* is TRUE. If we find any, we increment the number of constituent data packets in our transmission, and add the needed data to the transmission. At line 10 we finally send the transmission to the physical layer, and then start a timer to wait for the acknowledgment to come back from the receiver.

Figure 5-17 presents pseudocode for the sender's processing of feedback traffic from the receiver in lines 1–11. When the sender receives feedback from the receiver, it walks through the feedback message, looking up the

```
PP-ARQ-SEND (send_bytes)
1  WAIT(send_semaphore) ▷ Decrements send_semaphore by one
2  LPS ← LPS+1
3  bufent.data ← send_bytes
4  bufent.pkt_seqno ← LPS
5  bufent.rts ← TRUE
6  bufent.txcount ← 0
7  sendbuf.ENQUEUE(bufent)
8  if send_timer.RUNNING() = FALSE
9     then RESUME-XMIT-QUEUE()

RESUME-XMIT-QUEUE()
 1  tx.txcount ← 0 ▷ See Figure 5-12
 2  for bufent ∈ sendbuf.ELEMENTS()
 3       do if bufent.rts = TRUE
 4             then tx.txcount ← m.txcount+1
 5                  if bufent.txcount > 0
 6                    then tx[tx.txcount] ← GET-CHUNKS (bufent.data,
                               bufent.bad_chunks)
 7                    else  tx[tx.txcount] ← bufent.data ▷ First transmit
 8                  msg.rts ← FALSE
 9  if tx.txcount > 0
10     then SEND(Type-Data, tx)
11          send_timer.START (SEND-TIMEOUT, ACK_TIMEOUT)

SEND-TIMEOUT()
1  RESUME-XMIT-QUEUE()
```

FIGURE 5-16—Pseudocode for PP-ARQ's sending functionality. This procedure is executed by the higher layer at the sender only, and implements PP-ARQ, the sliding window, and selective, streaming acknowledgments.

```
PP-ARQ-RECEIVE (m)
 1  if m.type = TYPE-FEEDBACK ▷ Sender-side functionality
 2     then for i ∈ [1, m.fbcount] ▷ See Figure 5-13, p. 101
 3             do for chunks ∈ m.fb[i] ▷ See Figure 5-3, p. 93
 4                   do qentry ← sendq.LOOKUP(m.fb[i].pkt_seqno)
 5                      qentry.bad_chunks ← m.fb[i].bad_chunks
 6                      if LENGTH(qentry.bad_chunks) = 0
 7                        then qentry.acked = TRUE
 8                             SIGNAL(send_semaphore)
 9                        else qentry.rts ← TRUE
10         send_timer.STOP()
11         RESUME-XMIT-QUEUE()
12  elseif m.type = TYPE-DATA ▷ Receiver-side functionality
13     then fbpkt.fbcount ← 0
14          for i ∈ [1, m.txcount] ▷ See Figure 5-12, p. 100
15             do if nfe ≤ m.tx[i].seqno ≤ lfa
16                   then fbpkt.fbcount ← fbpkt.fbcount + 1
17                        qentry ← recvbuf.LOOKUP(m.tx[i].seqno)
18                        COMBINE(qentry.data, m.tx[i].data)
19                        (Λ^b, Λ^g, R) ← GEN-RUNLENGTH-REP (qentry.data)
20                        (C, Π) = BOTTOM-UP-PP-ARQ-DP(Λ^b, Λ^g, R, κ)
21                        chunks = GEN-FEEDBACK (Π, 1, R)
                          ▷ Fill in feedback; see Figure 5-3, p. 93
22                        fbpkt.fb[i].chkcnt ← LENGTH(chunks)
23                        fbpkt.fb[i].chunks ← chunks
24                        if CHECKS(chunks) and nfe = m.tx[i].seqno
25                          then NFE ← NFE + 1
26                               LFA ← LFA + 1
27                               recvbuf.REMOVE(m.tx[i].seqno)
28          if fbpkt.fbcount > 0
29            then SEND (TYPE-FEEDBACK, fbpkt)
```

FIGURE 5-17—Pseudocode for PP-ARQ's receiving functionality, at both the sender (receives feedback messages) and receiver (receives forward-link data messages). Together with PP-ARQ-SEND in Figure 5-16, this code implements the PP-ARQ streaming acknowledgment protocol.

send buffer entry corresponding to each constituent chunk. For each relevant send buffer entry, the sender marks the bad chunks as such, incrementing the SEND_SEMAPHORE if a packet has finished transmission.

Receiver-side protocol. Figure 5-17 presents pseudocode for the receiver's functionality in lines 12–29. The receiver maintains a buffer of packets "in-flight" between sender and receiver called *recvbuf*. Then starting at the loop at line 14, the receiver parses the forward-link transmission, looking up each fragment in the transmission (line 17), combining it with the packet data in its receive buffer, and then running BOTTOM-UP-PP-ARQ and associated helper functions (lines 18–21). The receiver then updates its variables *NFE* (next frame expected) and *LFA* (last frame acknowledged) at lines 25 and 26 if the combined packet meets checksum (tested by the CHECKS procedure at line 24).

5.4 IMPLEMENTATION

In this section, we describe our PP-ARQ implementation as well as the software infrastructure we have designed and developed to support our evaluation. We also point the reader to Section 3.3.1, p. 65 for a discussion of our Zigbee SoftPHY implementation, which comprises approximately 1,500 lines of C++ code and is also a part of the following experiments in Section 5.5.

Each Zigbee sender is a telos sensor network "mote" with a Chipcon CC2420 radio [117]. Senders run TinyOS[2] on the telos's TI MSP430 microprocessor. The CC2420 radio is a 2.4 GHz single-chip RF transceiver that uses direct-sequence spread spectrum (DSSS) at a bit rate of 250 Kbits/s (as described in §3.2.2, p. 57).

Each of the Zigbee receivers is a computer connected to a software defined radio. The hardware portion of the receiver is a Universal Software Radio Peripheral (USRP) [25] with a 2.4 GHz daughterboard; the remainder of the receiver's functionality (demodulation and block decoding as described in §3.2.2) is implemented in software. The DSSS despreading function is approximately 1,500 lines of original code written in C++ in the GNURadio [39] framework by the author, with parts derived from code by Schmid [106, 107].

---

[2] See `http://tinyos.net`.

```
(1!8,5) (c!e,2) (0!2,5) (8!c,10) (1!c,4) (b!1,10) (c!3,11)
(3!4,9) (9+,10) (e+,3)  (0+,0)   (4+,1)  (a+,0)   (1+,0)
(e+,0)  (9+,0)  (5+,0)  (2+,0)   (0+,0)  (5+,0)   (7+,0)
```

FIGURE 5-18—An excerpt from a Zigbee codeword-level packet trace from our SoftPHY implementation (see §3.3.1, p. 65). Each pair is of the form (data, Hamming distance). KEY TO SYMBOLS—!: incorrect data (followed by correct data); +: correct data.

PP-ARQ implementation. We have implemented PP-ARQ in 1,051 lines of Perl code, 610 of which implement the dynamic programming algorithm described in Section 5.2, and the remainder of which implement the PP-ARQ protocol described in Sections 5.1 and 5.3. Note that we did not implement the sliding window algorithm [93] described in Section 5.3 as such, but we did instrument the PP-ARQ code to keep track of the header overheads associated with each packet in order to correctly compute what those overheads would have been under the sliding window algorithm.

Reference scheme implementation. We have implemented each of the "reference point" schemes (described below in §5.5.2) in Perl code, totaling approximately 600 lines of code.

Trace-driven simulation implementation. Parts of our evaluation in Section 5.5 consist of trace-driven simulation results. The traces employed for these simulations are collected using our Zigbee SoftPHY implementation, which consists of approximately 1,500 lines of C++ code (see §3.3.1, p. 65 for a description of the Zigbee SoftPHY implementation). Conceptually, the traces are taken at the SoftPHY interface, which implies that they are taken at the Zigbee 4-bit codeword granularity. For every 4-bit data nibble, the Zigbee SoftPHY interface returns a corresponding integer between zero and 15 indicating confidence. Therefore the trace consists of a sequence of (4-bit data, 4-bit confidence) pairs, grouped into received frames.

Figure 5-18 shows an excerpt from one such trace. Each pair shows first the received data, followed by correct data if the received data is incorrect. The second tuple of the pair is a raw Hamming distance of the received data to the closest codeword, and is used to compute the associated Zigbee SoftPHY hint (see Equation 3.4, p. 56). We use the data in these traces to drive the PP-ARQ simulation and other protocol simulations indicated below.

Postamble based synchronization implementation. In addition to passing up SoftPHY hints, the physical layer implementation described in Section 3.3.1 performs postamble detection and synchronization using the "rollback" algorithm described in Section 4.4.1 (see in particular Figure 4-8, p. 82). For each frame received, the physical layer tags it with one bit that indicates whether the frame was detected via the preamble, or whether the preamble was undetectable and the frame was detected via the postamble. This "postamble" bit is available to our protocol simulations.

5.5 EVALUATION OF PP-ARQ

In this section we present a cumulative experimental evaluation of the three main contributions of this work: PP-ARQ with streaming, the SoftPHY interface, and the postamble mechanism. The main result of this section is an evaluation of aggregate throughput improvements that these contributions offer in a busy network where frequent collisions cause SINR to fluctuate.

Section overview. We begin our evaluation with a description of the overall experimental design in Section 5.5.1. Then in Section 5.5.2 we propose a set of goals for our evaluation, which includes three protocols that serve as meaningful reference points of PP-ARQ's performance. Next, to motivate the need for the flexible error recovery that PP-ARQ provides, we study the lengths and placements of errors in our testbed in Section 5.5.3 We present an end-to-end aggregate throughput performance evaluation in Section 5.5.4 and a study of the complexity of PP-ARQ at the receiver in Section 5.5.7.

We summarize our experimental contributions in Table 5.1. The top portion of the table summarizes and references the major experimental contributions of previous chapters, and the bottom portion of the table summarizes the experimental contributions made in the present chapter.

5.5.1 Overall experimental design

In our testbed, we have deployed 25 Zigbee sender nodes (as described in §5.4) over eleven rooms in an indoor office environment, as shown in Figure 5-19. We have also deployed six receivers (described in the same section) among the senders. In the absence of any other traffic, each receiver can hear between four and ten sender nodes, with the best links having near perfect delivery rates. All 25 senders transmit packets containing a known pseudorandom test pattern, at a constant rate, with a randomized jitter. The

Experiment	Radio	Section	Page	Result
802.11a OFDM SoftPHY hints	①	§3.2.3	p. 64	A proof-of-concept showing SoftPHY hints during one packet collision in a USRP-based OFDM receiver with the same structure as IEEE 802.11a.
SoftPHY hints in a busy network	③	3.3.1	65	SoftPHY hints have good predictive power of bit correctness in a busy, multi-hop network.
SoftPHY hints at marginal SNR	②	3.3.2	68	SoftPHY hints have best predictive power at high SNRs (BER less than 10^{-6} in radio system ②) but well for BERs as high as 10^{-3}.
PP-ARQ aggregate throughput	③	5.5.4	114	PP-ARQ improves aggregate and per-link end-to-end throughput by a factor of 2.1× in a busy, multihop base station topology.
PP-ARQ efficiency	③	5.5.5	115	PP-ARQ is 80% efficient compared to a hypothetical protocol that has a priori perfect knowledge of which bits are correct, and no overhead in requesting incorrect bits.
PP-ARQ computational requirement	③	5.5.7	120	PP-ARQ requires a very small amount of dynamic programming computation for most packets.
PP-ARQ v. Fragmentation	③	5.5.8	121	Under workloads with large packets, PP-ARQ improves performance by more than 30% over an optimally hand-tuned implementation of 802.11 fragmentation.

KEY—①: rate-1/2 convolutionally-coded OFDM (same structure as IEEE 802.11a); ②: uncoded QPSK; ③: direct sequence spread spectrum MSK.

TABLE 5.1—A roadmap of the major experimental results contained in this dissertation. *Above:* experimental results pertaining to SoftPHY hints and postamble decoding. *Below:* experimental results pertaining to PP-ARQ.

FIGURE 5-19—Experimental Zigbee testbed layout: there are 31 nodes in total, spread over 11 rooms in an indoor office environment. Each unnumbered dot indicates one of 25 Zigbee nodes. The six software defined radio nodes are shown dark, labeled with numbers.

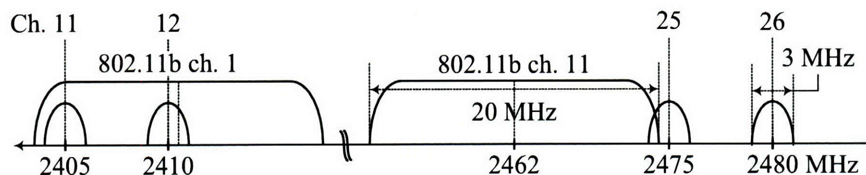


FIGURE 5-20—Selected Zigbee and WiFi channels in the 2.4 GHz ITU-R ISM frequency band. We evaluate our proposals in Zigbee channels 11 and 26 with both high and low background traffic volumes, respectively (figure not drawn to scale).

senders transmit at the same time, offering 6.9 Kbits/s/node unless otherwise noted. This represents 2.8% of the raw link bandwidth of 250 Kbits/s, or between 11%–28% of the raw link bandwidth at a given receiver.

Ambient RF environment. Our experimental results summarize data from Zigbee channel 11 at 2.405 GHz as shown in Figure 5-20. From the figure we note that Zigbee channel 11 overlaps with IEEE 802.11b channel 1, which carries active WiFi traffic in our building. Thus the experimental results we report below and in Chapter 5 were obtained in the presence of significant background traffic. We also validated our experimental results on Zigbee channel 26, with identical outcomes. Zigbee channel 26 overlaps with no IEEE 802.11 channels, so we expect it to be much quieter than channel 11. We used GNU Radio tools [39] to verify that there was indeed a high level of background traffic on Zigbee channel 11 and indeed significantly less background traffic on Zigbee channel 26.

Statistical methodology. All data points represent averages of 14 experimental runs, and all error bars indicate confidence intervals at the 95% level, unless otherwise noted. We use Student's t-test to compute the confidence intervals, thus non-overlapping confidence intervals allow us to conclude that an experimentally-measured difference is statistically significant.

5.5.2 Reference points for evaluation

We choose the following three protocols as meaningful reference points to evaluate PP-ARQ against. For each, we describe the protocol and explain its significance.

Packet checksum with ARQ

This scheme represents the status quo in IEEE 802.11a [52], 802.11b [51], and 802.11g [53]-compliant commodity wireless local area networking equipment, as well as IEEE 802.15.4 [54] sensor networking radios. Each transmitted packet contains a 32-bit checksum appended to the end. Postamble detection is turned off at the receiver.

For each incoming packet, the receiver computes the 32-bit CRC check over the received packet payload and sends an acknowledgment packet if it matches. Otherwise, the transmitter retransmits the packet in its entirety. We call this scheme *Packet CRC* in the following.

Fragmented checksum with selective ARQ

We will show next that PP-ARQ improves performance significantly, but one might ask whether it is necessary to achieve similar gains. One way to approximate PP-ARQ is to adopt a technique similar to that proposed in the IEEE 802.11 specification [50], splitting the packet into fragments whose size is user-tunable, and sending multiple checksums per packet, one per fragment, as shown in Figure 5-21. We call this scheme *Fragmented CRC* in the following.

For each incoming packet detected by the preamble, the receiver computes a 32-bit checksum over each fragment f and compares each to the corresponding received checksum appended to f. Fragmented CRC delivers only those fragments with matching checksums, discarding the remainder. In the acknowledgment packet, there is a bitmap indicating the result of each fragment's checksum verification.

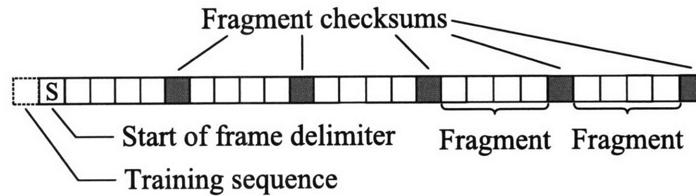


FIGURE 5-21—The per-fragment checksum approach: the packet includes multiple checksums, with each checksum taken over a different fragment of the packet.

This scheme allows the receiver to identify individual fragments that are correct. If bit errors are concentrated in only a few bursts, then entire fragments will checksum correctly, and the receiver would then only have to recover the erroneous fragments from the sender. Ganti et al. [34] provide a scheme to this effect.

SoftPHY best

In order to evaluate PP-ARQ, we wish to get a sense of how well it is performing in absolute terms. To this end we can ask the following question. *How much end-to-end bits-per-second throughput does PP-ARQ deliver, as a fraction of the rate that correct bits are being received?*

The answer is a fraction that quantifies how *efficient* PP-ARQ is, compared to a hypothetical protocol that has the following three properties. First, the receiver has *a priori*, perfectly-correct knowledge of which bits were received correctly and which were received incorrectly. Second, the receiver can communicate to the sender (without any overhead) exactly which bits were received incorrectly. And finally, there are no preamble or postamble overheads, yet the receiver can detect the presence of packets using either.

5.5.3 Error patterns in the Zigbee testbed

In these experiments, we seek an understanding of the basic statistics underlying the bit errors that our wireless testbed experiences. These figures further motivate the need for PP-ARQ, and help in explaining some of the performance trends we see later.

The length of bit errors. The basic design in this experiment is as given in Section 5.5.1. For each received packet in an experiment run on that setup, we check its known, pseudorandom payload, and measure the length l of

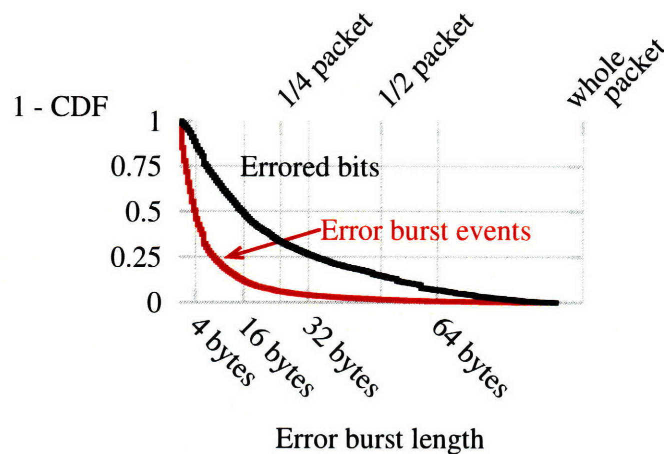


FIGURE 5-22—Burst error statistics in our Zigbee testbed. The (lower) curve labeled "error burst events" is the complementary cumulative distribution of error burst length. The (upper) curve labeled "errored bits" gives the fraction of bits that come from an error burst of at least the length shown on the ordinate axis.

each error burst i contained in every received packet, aggregating the results across multiple receivers.

The curve labeled "Error burst events" in Figure 5-22 shows the complementary CDF of random variable l, burst length. We see from the data that most error bursts in our Zigbee experiments are small, with half between one nibble and four bytes, and one of the remaining quarters between four and eight bytes.

Using the statistics of the burst length variable, we proceed to answer the different but related question: *which error burst lengths are responsible for which fraction of the errored bits?* Answering this question informs our design and evaluation of PP-ARQ. For example, if all the errored bits were of very short length (at most 1–2 bytes), then the dynamic programming algorithm that PP-ARQ uses would be heavily burdened from the many short runs of "bad" and "good" bits, and would also tend to request many chunks of the form $c_{i,j}$ where $i < j$ (for example chunks $c_{1,2}$ and $c_{2,3}$ in Figure 5-8, p. 96), retransmitting the correctly-received bits contained within.

To make the question precise, suppose that there are n_l error bursts of length l in the entire experiment. Then the fraction of errored bits coming

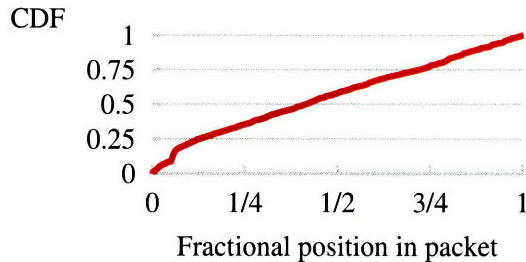


FIGURE 5-23—Experimental distribution of bit error locations.

from error bursts of greater than length j is

$$f(j) = \frac{\sum_{i=j+1}^{\infty} i \cdot n_i}{\sum_{i=0}^{\infty} i \cdot n_i}. \tag{5.7}$$

Performing the computation in Equation 5.7 results in the upper curve shown in Figure 5-22, labeled "errored bits." From the data and the data labels in the figure, we can directly answer the key question posed earlier in this section as follows.

1. Half of all errored bits come from burst errors of length 16 bytes or less,

2. one quarter of all errored bits come from burst errors of length 16–32 bytes, and

3. the remaining bit errors are of length 32 bytes (about one quarter packet length), with a rather long tail tending towards the entire packet length.

From these experimental answers, we can conclude that we should spend an equal amount of effort correcting bit error runs of length less than one quarter packet as we spend correcting bit error runs of greater than one quarter packet in length.

The location of bit errors. In the case of the Zigbee physical layer we use for our experiments, should we expect to dedicate more effort correcting errors in some parts of the packet rather than others? Again utilizing the basic experimental design of Section 5.5.1, we form a dataset containing the starting offset of every bit error run. From this dataset, we generate the cumulative distribution function of starting bit error locations, shown in Figure 5-23. We see that bit errors are uniformly distributed across the packet's length.

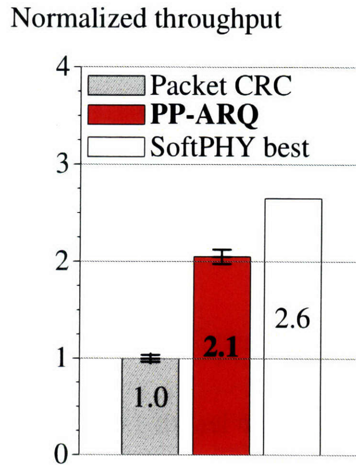


FIGURE 5-24—Aggregate throughput comparison between Packet CRC (the status quo), PP-ARQ, and an idealized protocol that delivers all correct symbols up to higher layers with no retransmissions nor overheads. PP-ARQ achieves a 2.1× speedup over the status quo.

These two results suggest that PP-ARQ will perform well, because it is flexible with respect to both varying error lengths, and error locations.

5.5.4 "Client-AP" aggregate network throughput

We now measure the aggregate network throughput that all nodes achieve in a "client-access point" topology where each client (unnumbered node in the network testbed shown in Figure 5-19, p. 109) picks the access point (numbered node in the same figure) to which it has the best frame delivery rate.

We design the experiment as follows. Using symbol-level data from the testbed SoftPHY packet traces (see §5.4, p. 106) collected from the testbed, we run our PP-ARQ protocol implementation (see §5.4, p. 106). Using the same traces, we also run Perl implementations of the Packet CRC (§5.5.2, p. 110) and SoftPHY Best (§5.5.2, p. 111) schemes, and measure the throughput that each achieves. Since these protocols involve bidirectional communication, we make the assumption that links are roughly symmetrical, and use the same sender-receiver packet trace to drive both directions of the protocol communication. De Couto et al. validate this assumption to a large degree: they find that the best third of all links in their wireless testbed have an average difference between their forward and reverse

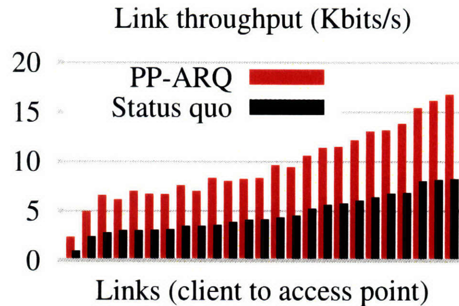


FIGURE 5-25—Per-link PP-ARQ v. status quo absolute throughput. PP-ARQ consistently doubles the reliable throughput achieved across each link in the network, relative to the status quo.

direction packet delivery rates of only approximately 5%, with a maximum difference of approximately 10% [20, §2.4].

Figure 5-24 shows the aggregate throughput of all nodes running PP-ARQ and "SoftPHY best," normalized to the aggregate throughput of all nodes using "Packet CRC." From the figure we see that PP-ARQ achieves a 2.1× speedup over "Packet CRC," and is a normalized factor of 0.5× slower than "SoftPHY best." In the next section, we investigate which mechanisms explain the 0.5× gap between PP-ARQ and "SoftPHY best," and which mechanisms explain PP-ARQ's performance improvement.

When we measure the throughput that PP-ARQ achieves over each link, we find that PP-ARQ's speedup applies evenly across all links, with links getting a benefit proportional to their status quo throughput; figure 5-25 shows this trend.

5.5.5 PP-ARQ efficiency

Taking the ratio of PP-ARQ and "SoftPHY best" performance figures from Figure 5-24 yields an efficiency of 81%. We now determine which factors account for this 19% overhead.

Feedback data overhead

One novel contribution of PP-ARQ is its use of the feedback channel to convey information about which bits the receiver has correctly used. This use of the feedback channel has overhead, which we quantify by measuring the

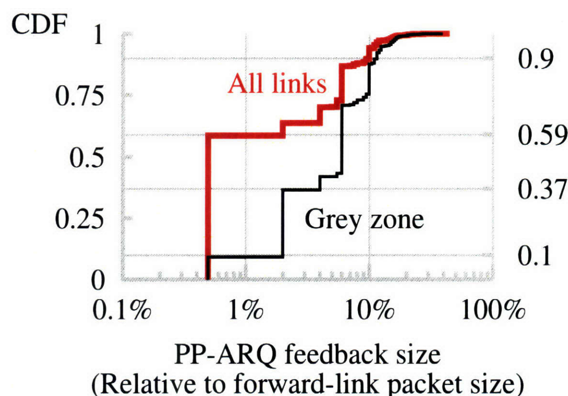


FIGURE 5-26—Distribution of reverse-link per-packet PP-ARQ feedback data sizes across all links (upper curve), and across only links in the "grey zone," which have a packet delivery rate of less than 70%.

number of feedback bits sent in the reverse direction (not including preamble and postamble bits), and expressing that figure as a fraction of the total bits sent in both directions (including preamble and postamble overhead):

$$\frac{\text{Feedback bits sent}}{\text{Total bits sent (both directions)}} = 4.5\% \tag{5.8}$$

This figure represents the percentage overhead that the feedback data alone (not including preambles and postambles) adds. While it is small in the aggregate, we note that in the "grey zone" where link delivery rates are less than 70%, this figure jumps to 29%.

To see why PP-ARQ feedback data overhead increases in the grey zone, we measure the size of each feedback data payload. Figure 5-26 shows the resulting distribution of the feedback data sizes. Looking at the curve marked "all links", we see that across all links in the experiment, 59% of the feedback data fragments (see §5.1.1, p. 92) were 0.5% of the forward-link packet size, 101 bytes. These feedback data correspond to cases where the forward-link transmission was received correctly in its entirety, and the only feedback data necessary is the header information in the feedback fragment identifying the transmissions sequence number (see Figure 5-3, p. 93). There is another mode in the feedback size distribution at 6% of the forward-link packet size corresponding to roughly 30% of transmissions, and then a small tail (10% of transmissions) with significant (greater than 10% of

forward-link packet size) length. These larger feedback packets indicate more complicated bit error patterns in the received forward-link packets.

We plot the feedback size distribution in the grey zone with the curve in Figure 5-26 marked "grey zone." The tail of the "grey zone" distribution (feedback lengths above 10%) almost exactly matches the tail of the "all links" distribution, suggesting that the worst links have the most feedback overhead (a per-transmission overhead greater than 10% of the forward-link packet size). Furthermore, the "grey zone" curve mirrors the "all links" curve, with a downward shift caused by fewer small (0.5% forward-link size) feedback transmissions. This further suggests that the small feedback transmissions should be attributed to the best links. We conclude that worse links cause PP-ARQ to send more feedback.

PP-ARQ specific headers

In order to parse forward-link transmissions as described in Section 5.1, the PP-ARQ protocol reserves space in each data transmission for headers (Figure 5-4, p. 93).

$$\frac{\text{PP-ARQ specific header bits sent}}{\text{Total bits sent (both directions)}} = 0.68\% \tag{5.9}$$

Discarding bits labeled "good"

When the receiver labels a run of bits "good," it requests transmission of a checksum of those bits (see §5.1, p. 88). When it receives the checksum (itself labeled "good"), it then computes the checksum over the received run of good bits, and verifies the result against the received checksum. If the verification fails, then the received discards the "good"' run of bits, marking them all "bad" and continuing the PP-ARQ protocol.

We measure how many bits in our experiments are marked discarded in this way, as a fraction of the total bits sent in both directions:

$$\frac{\text{Bits discarded}}{\text{Total bits sent (both directions)}} = 0.35\% \tag{5.10}$$

Choosing to resend "good" bits

In our discussion of PP-ARQ's dynamic programming (DP) in Section 5.2, we noted that the DP algorithm often chooses to aggregate a request for two or more "bad" runs of bits into one request for a run of bits that includes

Source of overhead	Overhead	
	Pct. of total	Normalized factor
Preambles/postambles	17%	0.44×
Feedback channel data	4.5	0.1
PP-ARQ specific headers	0.68	0
Discarding bits labeled "good"	0.35	0
Choosing to resend "good" bits	0.25	0
Total	23	0.6

TABLE 5.2—A summary of the overhead sources and impacts that PP-ARQ incurs. The second column of the table tabulates the overhead as a percentage of the total bits transmitted and received. The third column tabulates the overhead as an improvement factor normalized to the performance of the Packet CRC scheme. "PP-ARQ specific headers" refers to the shaded headers in Figures 5-3, 5-5, and 5-6 on pages 93–94.

some "good" bits. For example, PP-ARQ chooses chunk $c_{1,2}$ in Figure 5-8 (p. 96), retransmitting the "good" bits contained in that chunk.

We measure how many bits in our experiments PP-ARQ chooses to retransmit, as a fraction of the total bits sent in both directions:

$$\frac{\text{"Good" bits intentionally resent}}{\text{Total bits sent (both directions)}} = 0.25\% \tag{5.11}$$

★ ★ ★

We summarize the different sources of overhead in Table 5.2. All together, the sources of overhead sum to 23% of the overall bits transmitted and received, or a relative factor of 0.6×. Looking back at Figure 5-24, we see that the computed performance of "SoftPHY optimal" minus this overhead factor is within the 95% confidence interval of PP-ARQ's end-to-end performance. Thus, the sources of overhead listed in Table 5.2 explain the performance gap between PP-ARQ and "SoftPHY optimal."

5.5.6 PP-ARQ improvement

We now investigate the reason for PP-ARQ's improvement over the status quo. Every time a bit error occurs in the "Packet CRC" scheme, it has to retransmit the entire packet, wasting many bits. Figure 5-27 shows the cumulative distribution of retransmission sizes for the two schemes. The curve

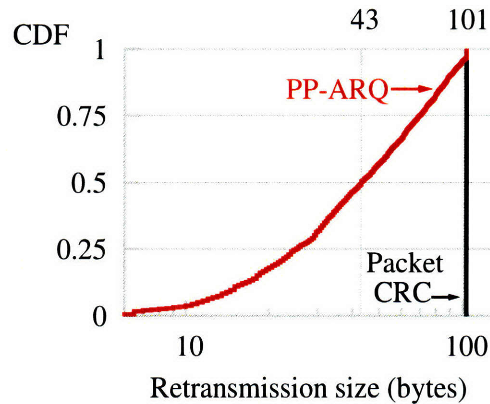


FIGURE 5-27—Distribution of forward-link retransmission sizes for each of the two protocols we have evaluated thus far. PP-ARQ more than halves the median status quo retransmission size.

labeled Packet CRC on the right shows a fixed distribution of 101 bytes accordingly.

We note also that the cumulative distribution of PP-ARQ's retransmission size requests in Figure 5-27 almost exactly mirrors the complementary cumulative distribution of burst error lengths in Figure 5-22 (p. 112). This is expected because PP-ARQ requests not too much more than those bits which are in error.

Isolating the performance contribution of the postamble

In Section 5.5.4 we compared the performance of PP-ARQ, which uses the postamble as well as the preamble for packet detection, with the status quo, which does not. This motivates the question of how much of PP-ARQ's gains are due to the postamble, and how much are due to protocol operation.

Using our SoftPHY packet traces (§5.4, p. 106), we measure the expected number of transmissions (ETX [20]) required to deliver one packet across each link, with packets detected via the postamble included. Then we repeat the same measurement for each link, with packets detected via the postamble excluded.

In Figure 5-28 we show two views of a scatter plot with one data point per link, placed at the two ETX values computed. Over each link, preamble+postamble detection gives gains of 0%–33% over preamble-only de-

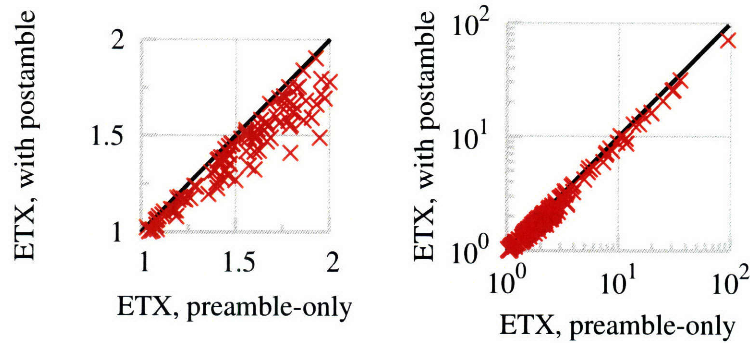


FIGURE 5-28—Postamble detection improves the throughput of many links in the "SoftPHY ideal" scheme, relative to the same scheme using only preamble detection. *Left*: linear scale detail; *right*: logarithmic scale.

tection. Therefore we attribute most of PP-ARQ's improvement in Section 5.5.4 to PP-ARQ's protocol operation.

5.5.7 PP-ARQ receiver timing issues

When the SoftPHY-enabled physical layer receives a packet and passes it up to the PP-ARQ algorithm (also running at the receiver), the PP-ARQ dynamic programming algorithm needs to run at the receiver before the receiver can begin to transmit the feedback packet. The time in between these two events is usually (depending on the details of the medium access control protocol) wasted. For example, IEEE 802.11 [50] gives the receiver of a transmission priority by forcing nodes within range of the transmission to defer by a fixed time interval.

One mitigating factor is that like other demanding baseband processing that may take place at the receiver (Viterbi decoding of long constraint-length codes, for example), the PP-ARQ DP computation can be performed at the same time the radio is switching from receive to transmit mode (a maximum of 2 μs in 802.11a, 5 μs in 802.11b/g). Nonetheless, we seek to show in this section that the amount of processing required is not prohibitive.

From our SoftPHY packet traces (§5.4, p. 106), we measure the number of runs R in each received packet, as defined in Equation 5.1 on page 90. Figure 5-29 shows the complementary CDF of R. The majority (80%) of

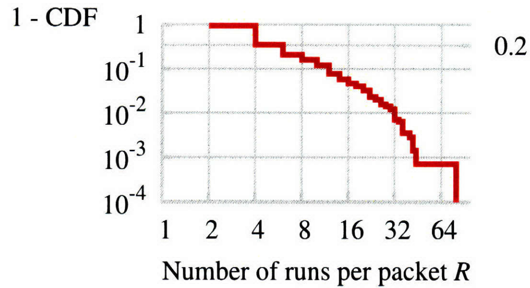


FIGURE 5-29—Distribution of the number of "runs" R received in each packet, as defined in Equation 5.1 on page 90.

received packets contain at most four runs, and 99 in 100 contain 32 or fewer runs.

5.5.8 Comparing against fragmented checksums

We now present results comparing the SoftPHY interface and PP-ARQ protocol to the fragmented checksum scheme, as described in Section 5.5.2 on page 110. Our results in this section are from trace-driven simulation, using the traces described in Section 5.4, on page 106. We simulate a range of different packet sizes realistic for a mesh network [110], attaching an IEEE 802.11-size preamble [50] to each packet.

We begin by tuning the fragment size of the fragmented checksum scheme, in order to compare it most favorably against SoftPHY and PP-ARQ.

Tuning the fragmented checksum scheme

Under the fragmented checksum scheme, how big should a fragment, c, be? In an implementation, one might place a checksum every c bits, where c varies in time. If the current value leads to a large number of contiguous error-free fragments, then c should be increased; otherwise, it should be reduced (or remain the same). Alternatively, one might observe the symbol error rate (or bit error rate), assume some model for how these errors occur, and derive an analytically optimal fragment size (which will change with time as the error rate changes). In either case, the fragmented checksum needs tuning for the optimal fragment size. In this section, we investigate the best case for per-fragment checksums, finding from traces of errored

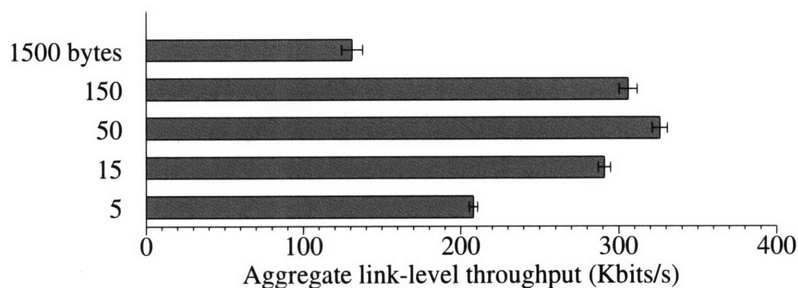


FIGURE 5-30—The impact of fragment size on the performance of the fragmented checksum scheme. Postamble decoding is off, carrier sense on in this experiment.

and error-free symbols what the *post facto* optimal fragment size is and using that value.

In these experiments, we post-process the traces to simulate a range of packet sizes realistic for the Internet [110] (this technique is accurate in the busy, collision-dominated network that we evaluate in this section).

To find the optimal chunk size for the fragmented CRC scheme, we compare aggregate throughput as fragment size varies. The results are shown in Figure 5-30. We see that when chunk size is small, checksum overhead dominates, since there are many 32-bit checksums in each packet. Conversely, large chunk sizes lose throughput because collisions and interference wipe out entire fragments. We therefore choose a fragment size at the sweet spot of 50 bytes (corresponding to 30 fragments per packet) for the following experiments.

SoftPHY unreliable throughput

To gain further insight about PP-ARQ's performance gains, we look one layer deeper, at the unreliable throughput achieved at the SoftPHY interface. These experiments measure *unreliable throughput*—correct bits per second received without any of the retransmission overheads involved in an ARQ protocol. Figure 5-31 compares the per-link distribution of throughputs at medium offered load (where each node transmits at a rate equal to 2.8% of the raw link speed) for each scheme. Since all bits in the packet share fate in the packet-level checksum scheme, performance with or without postamble decoding in that case is very close, and so for clarity we omit the curve for packet-level checksum with postamble decoding.

Looking at per-link throughput in Figure 5-31 that almost one quarter of all the links achieve no throughput in the status quo, because all of the

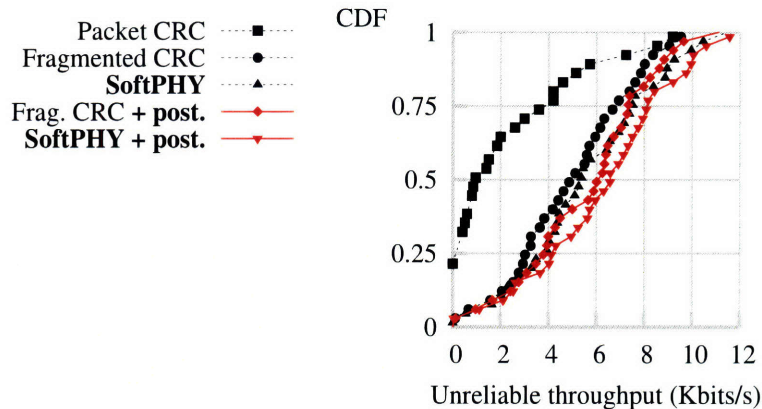


FIGURE 5-31—Per-link throughput distribution achieved at the SoftPHY interface. The per-node offered load is 2.8% of the link speed, close to channel saturation.

packets have some small number of bit errors. In contrast, the same lower quartile of links with SoftPHY and postamble decoding enabled are able to achieve up to four Kbits/s in unreliable throughput.

We also note that fragmented checksum yields a median 4× unreliable throughput gain over the status quo, and that SoftPHY yields median 7× unreliable throughput gain over the status quo, without the need for tuning the fragment size, as noted above.

Furthermore, postamble decoding (labeled "+ post." in Figure 5-31) yields another additive gain over either SoftPHY or Fragmented checksum in raw unreliable throughput, because these schemes can recover the correct bits from a packet whose preamble and/or body were undetectable and corrupted, respectively.

A link-by-link comparison. The scatter plot in Figure 5-32 compares unreliable throughput for fragmented CRC on the x-axis with either SoftPHY (top half) or packet-level CRC (bottom half). The first comparison we can draw from this graph is the per-link throughput of SoftPHY compared with fragmented CRC (top-half points). We see that SoftPHY improves per-link performance over fragmented CRC by roughly a constant factor. This factor is related to the fragment size, and may be attributable to fragmented CRC's need to discard the entire fragment when noise or another transmission corrupts part of it.

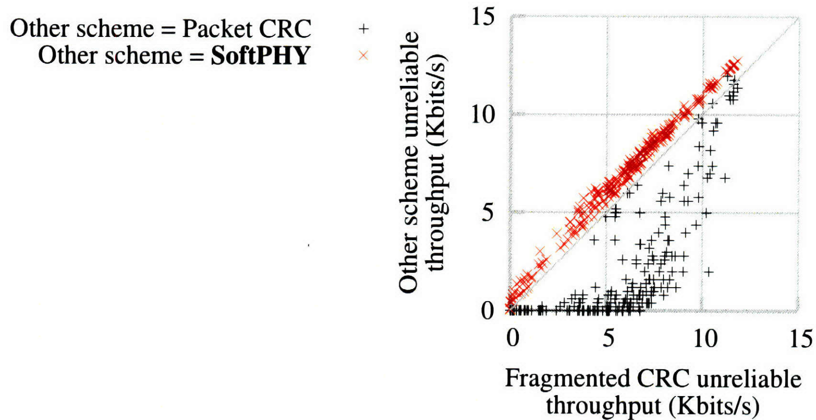


FIGURE 5-32—Link-by-link comparison of unreliable throughput at the physical layer boundary: each data point represents one link in one experimental run. The per-node offered load is 2.8% of the link speed, close to channel saturation. *Upper half*: SoftPHY v. fragmented CRC. *Lower half*: packet-level CRC v. fragmented CRC.

The bottom-half points in Figure 5-32 compare fragmented CRC with packet-level CRC. We see that fragmented CRC (and SoftPHY) far outperform packet CRC, because they only have to discard a small number of bits instead of the entire packet when those bits are corrupted. The fact that most of the lower-half points cluster near the x-axis means that the spread in the unreliable link throughput distribution increases when moving to smaller fragment sizes or SoftPHY. This is likely because collisions do not occur the entire packet, but rather occur over a small piece of it.

Equivalent frame delivery rate

We now examine the rate at which each scheme described above delivers correct bits to higher layers, once it has successfully acquired a packet (i.e., the physical layer has detected either a preamble or a postamble). We call this rate the *equivalent frame delivery rate*, because it measures how efficient each scheme is at delivering correct bits to higher layers once the physical layer successfully synchronizes.

Figure 5-33 shows the per-link distribution of equivalent frame delivery rate in our network when each node offers a moderate traffic load (2.8% of the raw link speed). Even when carrier sense and postamble decoding are

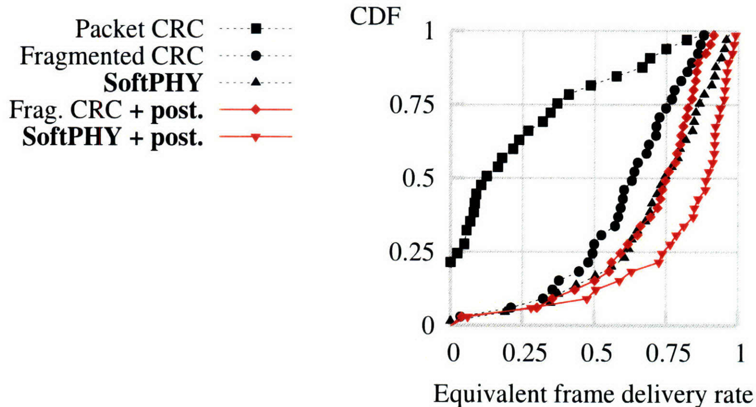


FIGURE 5-33—Per-link equivalent frame delivery rate distribution with carrier sense enabled, at moderate offered load (2.8% of the raw link rate, per node).

enabled, we see a large proportion of extremely poor links in the status quo network, labeled "Packet CRC" in the figure.

Comparing Figure 5-33 with Figure 5-31, notice that in both figures, just under 25% of links in the Packet CRC scheme attain no throughput and no frame delivery rate. Since Figure 5-33 measures bit delivery rate after successful synchronization, it rules out packet detection as the reason that 25% of links in Figure 5-31 attain no throughput.

Techniques for partial packet recovery increase frame delivery rate substantially, however. For both SoftPHY and the fragmented CRC scheme, postamble decoding increases median frame delivery rate by the fraction of bits that come from packets whose preamble was undetectable, roughly 10%. Comparing packet-level CRC with fragmented CRC, we see a large gain in frame delivery rates because fragmented CRC does not throw away the entire packet when it detects an error. The SoftPHY interface improves on frame delivery rates even more by identifying exactly which portions of the frame are correct and passing exactly those bits up.

Impact of carrier sense. We now repeat the experiment with carrier sense disabled; Figure 5-34 shows the results. When carrier sense is disabled, at least a small part of the packet is likely to be decoded incorrectly, resulting in a dropped packet in the packet-level CRC scheme. This is reflected in the very poor frame delivery rates of packet-level CRC. However, at moderate offered loads, we see that it is not likely that very much of the packet is

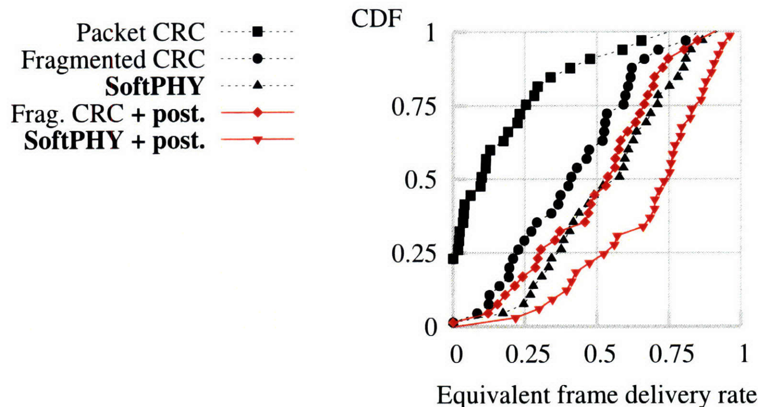


FIGURE 5-34—Equivalent frame delivery rate with carrier sense disabled, at at moderate offered load (2.8% of the raw link rate, per node). These experimental parameters is identical to those of Figure 5-33, except that carrier sense is disabled in these results.

involved in a collision, because the frame delivery rates for PPR and fragmented CRC remain roughly unchanged between Figures 5-33 and 5-34.

Impact of increasing load. We repeat the experiment with carrier sense enabled, but at a higher per-node offered load of 5.5% of the raw link bandwidth. Figure 5-35 shows the results.

PP-ARQ v. Fragmented checksums

Figure 5-36 shows the aggregate received throughput across all links in the testbed for packet-level CRC (the status quo), fragmented CRC, and PP-ARQ. We see that PP-ARQ achieves roughly a 2× capacity improvement over the status quo, **without** needing the fragment-size tuning described in Section 5.5.8.

One significant cause of our performance improvements over the status quo is the avoidance of retransmitting data that reached the receiver, but was discarded due to a bad checksum. Figure 5-37 quantifies this intuition. In the status quo ("Packet CRC" in the figure), retransmissions are always packet-sized in length, and so we see only the modes of the packet distribution in the retransmit-size distribution. Fragmented CRC tuned with a fragment size of 50 bytes breaks the retransmissions down into fragments

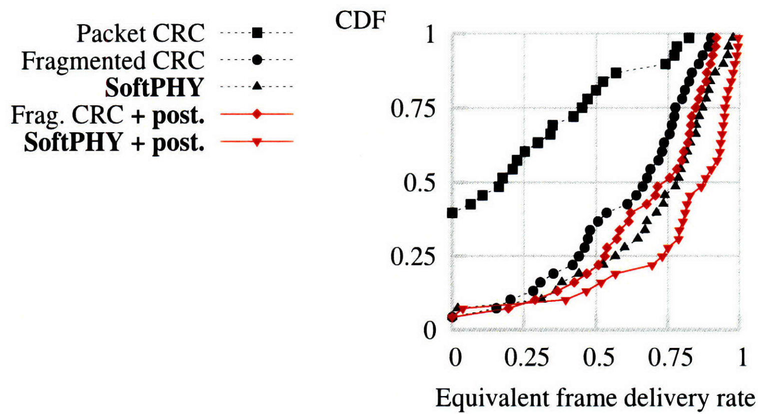


FIGURE 5-35—Equivalent frame delivery rate, with carrier sense enabled, at a high offered load (5.5% of the raw link rate, per node). These experimental parameters are identical to those of Figure 5-33 except that the per-node offered load is increased from 2.8% to 5.5% of the raw link bandwidth.

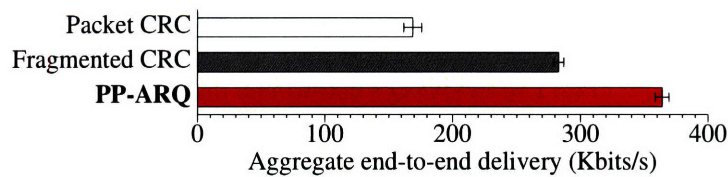


FIGURE 5-36—Comparison of the aggregate end-to-end delivery rate between packet-level CRC, fragmented CRC, and the PP-ARQ implementation. Postamble decoding is on in this experiment.

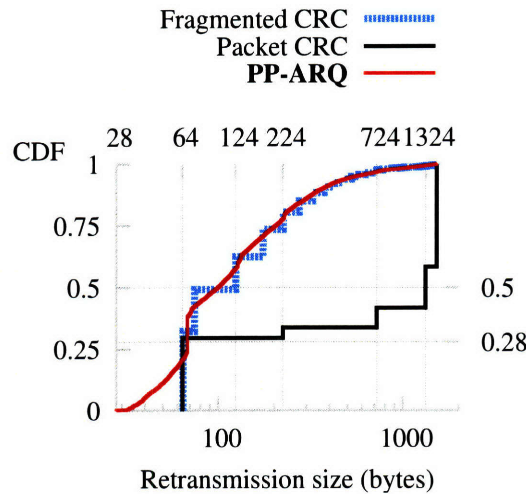


FIGURE 5-37—Comparison of the distribution of retransmission sizes for packet-level CRC, fragmented CRC, and the PP-ARQ implementation. Note PP-ARQ's long tail of short retransmit sizes.

of size $50 \times k$ for positive integers k, resulting in the stair-step pattern in the figure. However, fragmented CRC transmits no fragments smaller than 64 bytes. In contrast, PP-ARQ transmits a significant fraction of very small packets (less than 64 bytes), the cause of its significant performance gains. Note from Section 5.3 that PP-ARQ batches its retransmissions to avoid preamble overhead on each of the smaller retransmissions.

Figure 5-38 shows how end-to-end delivery rate changes when we increase the offered load to in the network. As well as raw offered load, we show the percentage of link capacity each node offers in the figure. At higher offered loads we see packet-level CRC performance degrading substantially. There have been several recent studies that attempt to elucidate the causes of this loss [2,112,113]. PP-ARQ's end-to-end throughput increases despite the overload, suggesting that only relatively-small parts of frames are actually being corrupted in overload conditions in the status quo.

The impact of carrier sense. In other work [123] and in Chapter 2 we have shown that selectively disabling carrier sense (see Chapter 2 for a review of how carrier sense works) can improve throughput.

One potentially confounding factor in our evaluation is the use and efficacy of carrier sense in the senders' CC2420 radios: carrier sense can fail due to hidden terminals or backoff slots smaller than the transmit-to-receive

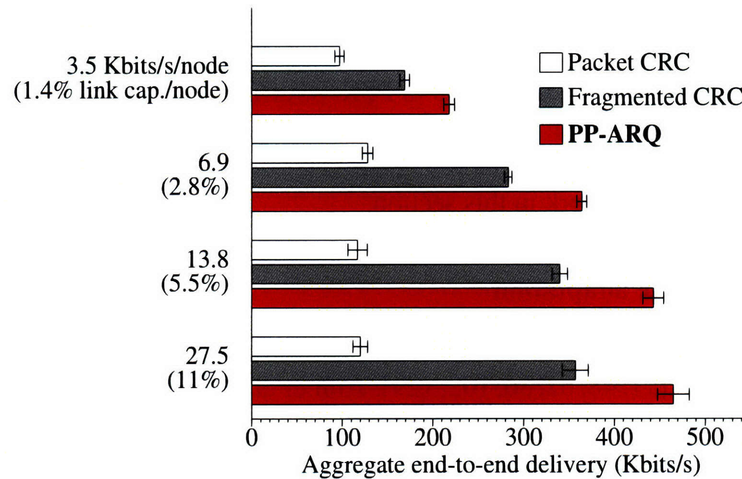


FIGURE 5-38—Comparison of end-to-end delivery rate in overload conditions; PP-ARQ scales favorably compared to the status quo. Postamble decoding is enabled in this experiment.

turnaround time [23]. To address this factor, we examine aggregate throughput for each scheme, with and without carrier sense. In Figure 5-39 we see that carrier sense improves throughput by a statistically significant amount over the status quo ("Packet CRC" with postamble decoding off). Noting that carrier sense yields additive improvements for each scheme, we narrow the design space of our evaluation to only include carrier sense on in the remaining experiments.

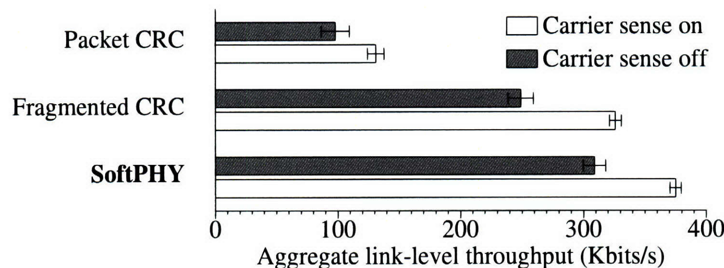


FIGURE 5-39—The impact of carrier sense on aggregate link-level throughput. Carrier sense improves throughput under each scheme, but PPR techniques yield further improvements. Postamble decoding is off in this experiment.

5.6 RELATED WORK

While each of the three ideas in PPR—SoftPHY, postamble decoding, and PP-ARQ—is novel, as is their synthesis into a single system, these individual ideas are related to and inspired by much previous work. We survey closely related work in this section.

5.6.1 Rate adaptation

Rate selection algorithms have been extensively studied in 802.11 wireless networks [11, 48, 62, 86, 102, 129]. Ahn et al. [3] propose an adaptive FEC algorithm which dynamically adjusts the amount of FEC coding per packet based on the presence or absence of receiver acknowledgments.

These algorithms work by examining frame loss statistics at the link layer, and directing the physical layer to increase or decrease the rate of the link according to some policy. The physical layer typically does this by changing the modulation on the link, or adjusting the amount of channel coding performed. In IEEE 802.11a and 802.11g, this is done by *puncturing* (removing bits in a systematic manner from) the output of the convolutional coder [97].

However, as a result of hidden interfering transmissions (see Chapter 2 for further discussion), it is extremely difficult to predict how much redundancy a wireless link will need in such highly-variable conditions. Bit rate adaptation algorithms take an empirical approach, commonly making "mistakes:" increasing the rate on a link until frames get dropped, and then decreasing the rate.

Our contributions impact bit-rate adaptation in several different ways. First, with the SoftPHY interface, much more information is available to the bit-rate adaptation algorithm, enabling better performance. PP-ARQ mitigates the penalty for choosing the incorrect rate by allowing receivers to recover partially-received frames and efficiently retransmit only the parts missing. We make further observations about bit-rate selection in general in the next chapter.

5.6.2 Incremental redundancy and hybrid ARQ

In general, the term "Hybrid ARQ" refers to any scheme that combines forward error correction and automatic repeat request. Type I hybrid ARQ schemes [70] retransmit the same coded data in response to receiver NACKs. Wireless local-area networking hardware supporting IEEE 802.11a

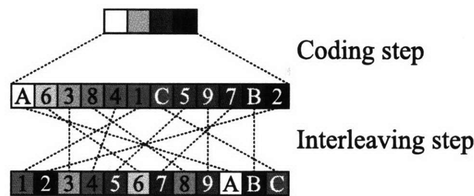


FIGURE 5-40—Coding and interleaving data in preparation for applying incremental redundancy in communications. Hexadecimal numbers refer to the order with which the constituent bit-blocks are transmitted. Shading indicates the packet position to which the data belongs.

[52], 802.11b [51], and 802.11g [53] as well as IEEE 802.15.4 [54] wireless sensornets all fall under this category.

Type II hybrid ARQ schemes [70] forego aggressive FEC while the channel is quiet, instead sending parity bits on retransmissions, a technique called incremental redundancy (IR) [73]. Metzner [78], Lin and Yu [71], and Soljanin et al. [111] have developed incremental redundancy hybrid ARQ schemes. IEEE 802.16e [55, §6.3.17], [56, §8.4.9.2.1.1, §8.4.9.7], the "WiMax" wireless metropolitan area networking uses incremental redundancy, as does the high-speed downlink packet access (HSDPA) third generation mobile telephony protocol for mobile broadband data delivery [47].

IR schemes combine coding with interleaving [6, Chp. 12] to spread the bursts of errors associated with collisions and deep fades across the entire packet. To illustrate the important concepts, we describe a simplified IR scheme based on a trivial rate-compatible punctured code [43].

Each packet is first coded at some rate R_c; here $R_c = 1/3$; this step is labeled "coding step" in Figure 5-40. Then, the sender interleaves the bits in the packet by some permutation known a priori to both sender and receiver; this is the "interleaving step" in the figure. The result of the coding step is to add more redundancy to the bits in the packet, and the result of the interleaving step is to locate information about non-neighboring regions of the original packet at consecutive bit locations in the interleaved packet. We show this schematically in Figure 5-40 by the degree of shading: light shading denotes information from the leftmost bit in the original packet, dark shading denotes information form the rightmost bit, with successive gradations denoting the position of bits in between.

After the coding and interleaving steps, the data are ready for transmission; we refer to these data as the "transmit data." The transmission process is shown in Figure 5-41. The sender first transmits one-third of the transmit

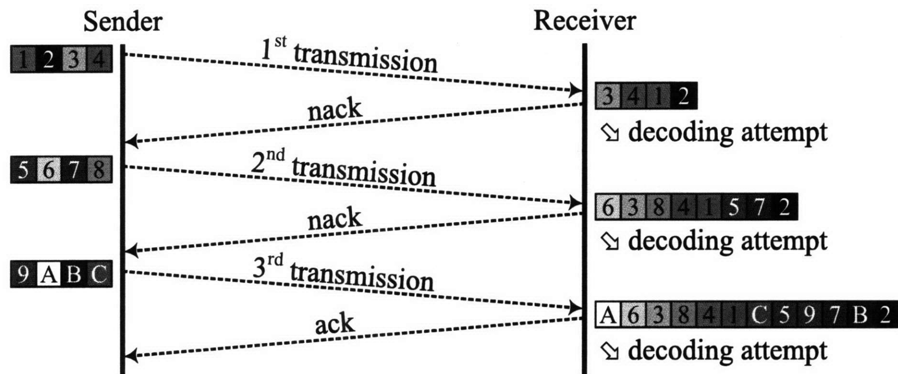


FIGURE 5-41—Incremental redundancy Hybrid ARQ. The sender applies some amount of channel coding as shown in Figure 5-40, and then sends a different set of interleaved bits in each successive transmission. One each transmission, the receiver makes a decoding attempt using the received transmission and all previous transmissions, sending "nacks" in response until it can decode the packet, at which point it sends "ack."

data over the air, where it may be corrupted; this is labeled "first transmission" in Figure 5-41. Upon receiving the first transmission, the receiver de-interleaves the data; the result is shown on the right-hand side of the figure. The receiver then makes a decoding attempt on the first transmission, typically using the Viterbi algorithm in the case of convolutionally-coded data. If the decoding attempt is successful (indicated by the successful verification of a packet-level checksum) then the receiver sends an "ack" to the transmitter and delivers the packet to the higher layer. Otherwise, it sends a "nack" (either via an explicit message or a simple timeout of an "ack" message). Upon receiving an "ack" message, the sender moves on to the next data packet; otherwise it sends the next third of the transmit bits to the sender. Upon receiving the second group of transmit bits, the sender combines the bits in both transmissions in its receive buffer (as shown at the terminating end of the second transmission in Figure 5-41) and makes another decoding attempt. The process continues until the sender runs out of encoding bits (which may be never in the general case of a rateless code), or the receiver makes a successful decoding attempt.

Comparing PP-ARQ with incremental redundancy

We first note that incremental redundancy (IR) alone enables none of the other uses of the SoftPHY interface discussed in the following chapter and

alluded to in Chapter 1. The pertinent comparison is therefore between PP-ARQ and IR. PP-ARQ takes a fundamentally different approach from IR: instead of coding and interleaving over the entire packet, it uses hints from the physical layer about which codewords are more likely to be in error, and retransmits just those codewords.

There are circumstances when the approach we advocate may outperform IR. For example, suppose that a burst of noise occurs on the channel of a length such that to successfully decode the packet, the receiver requires slightly more than one transmission's worth of data in Figure 5-41. The IR receiver will "nack" the first transmission, requiring the second before making a successful decoding attempt. In contrast, PP-ARQ will retransmit just enough bits to successfully reconstruct the packet, under the assumption that the underlying bit rate adaptation algorithm has selected the best bit rate. To address this drawback, IR could be modified in the vein of PP-ARQ to use the feedback channel for telling the sender how much more redundancy to transmit, instead of using fixed-size redundancy increments.

IR has the following two advantages over our approach. First, the interleaving step in Figure 5-40 and the corresponding deinterleaving step at the receiver spread bit errors out over the entire transmission, decoupling performance from the statistics of bit errors. In contrast, PP-ARQ's performance depends on the statistics of bit errors; in particular, PP-ARQ works best when errors occur in long bursts, making them easily encodable into descriptions of runs. Second, in the high-BER regime, PP-ARQ incurs significant feedback channel overhead, which IR avoids.

We leave the integration of PP-ARQ with bit rate adaptation and an experimental performance comparison between PP-ARQ and IR as future work. Another interesting open question is how the individual strengths of PP-ARQ and IR could be combined.

5.6.3 Packet combining strategies

PP-ARQ uses a simple replacement policy to combine successive transmissions. Also in the context of a single link, Chase combining [15] improves on this strategy by storing the multiple soft values of each received symbol in the packet and feeding them all to the decoder.

There are also many networked-systems designs in the literature that use packet combining in various contexts in which there is receiver diversity. Avudainayagam et al. [4] propose a scheme for exchanging soft information between several receivers of a packet that minimizes the overhead involved in the mutual sharing of information between the receivers. Woo et

al. [131] propose a system for combining symbols from multiple receptions of a packet at different wireless access points using a technique similar to maximal ratio combining [97] of the soft information of each symbol. We discuss this work in further detail in the next chapter. Dubois-Ferrière et al. [22] propose a system for combining packets in wireless mesh networks using hard decisions and a block coding strategy.

5.6.4 Soft-decision decoding

In short, soft-decision decoding (SDD) is a forward-error correction technique; SoftPHY is an expanded interface to the physical layer. SDD uses soft information to decode data transmitted over the channel with error-correction coding applied. Conventional SDD physical layers then pass up just the resulting bits using the status quo physical layer interface instead of the SoftPHY interface. While SDD improves bit error rates over individual links in isolation, all the problems associated with wireless network design noted in Chapter 1 still apply.

Note that we have implemented SoftPHY in conjunction with SDD in the 802.11a receiver design presented in Chapter 3. In cases like these, a soft-input, soft-output (SISO) soft-decision decoder uses SDD *and* passes up SoftPHY hints as described in that chapter.

5.6.5 Turbo coding

Turbo codes [8] combine two convolutional coders in parallel with an interleaver in between, which shuffles the bits input to one of the encoders. The turbo decoder operates in an iterative manner, passing the soft outputs from one decoder to the other in each iteration (for details, see Proakis [97]).

The use of coding and SoftPHY together is possible in most cases, because all the physical layer need provide is soft outputs from the outermost-layer decoder. This is not the case, however, in some Turbo decoders. There, successive iterations of Turbo decoding cause the soft information about each bit to "converge" to hard information, leaving no information to pass upwards via the SoftPHY interface. While this type of receiver has found a well-known use in the extremely low SNR regimes found in deep-space satellite communication [18], it is typically not used in the wireless networks we consider in this work, because communicating across very long (and hence low SINR) links reduces the capacity of the network as a whole [42].

6

Conclusion

THIS DISSERTATION has proposed two mechanisms that together define a new interface to the physical layer, the SoftPHY interface. The previous chapter proposed one use of the SoftPHY interface: increasing the reliable throughput achieved over a link. However, there are many other uses, which we explore in this chapter.

Chapter overview. We begin with a look at two proven uses of the SoftPHY interface in medium access control and network coding in Section 6.1. We then offer some final thoughts to in Section 6.2.

6.1 OTHER USES OF SOFTPHY

We now discuss three other uses of the SoftPHY interface besides increasing the reliable throughput over a wireless link. As these examples illustrate, SoftPHY has had impact on protocol design across three layers of the wireless networking stack.

6.1.1 Medium access control

It is well-known that maximizing the number of successful concurrent transmissions is a good way to maximize the aggregate throughput in a wireless network. Current contention-based channel access protocols generally attempt to minimize the number of packet collisions, allowing concurrent transmissions only when the nodes determine that they are unlikely to result in a collision. For example, in the popular *carrier sense multiple access*

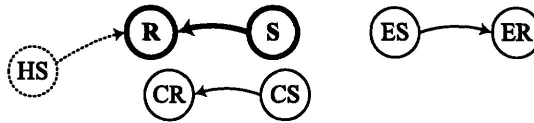


FIGURE 6-1—An example transmission from **S** to **R** with three abstract sender cases: an in-range but conflicting sender **CS**, an exposed sender **ES**, and a hidden sender **HS**.

(CSMA) scheme, before transmitting, a sender listens to the channel and assesses whether a nearby node is transmitting. If no nearby node is transmitting, the sender transmits immediately. If a nearby node is transmitting, then the sender defers, waiting for some time after the end of the on-going transmission. Then the sender repeats the same carrier sense–defer process.

Because a receiver's ability to decode a packet successfully depends on channel conditions near the *receiver* and not the sender, CSMA is at best a sender's crude guess about what the receiver perceives. This guess can be correct if the receiver and sender are close enough that they experience similar noise and interference conditions. However, it can also prevent a sender (e.g., **ES** in Figure 6-1) from transmitting a packet when its intended destination has a lower level of noise and interference—an *exposed terminal* situation. In addition, researchers have observed that receivers can often "capture" packets from a transmission even in the presence of interfering transmissions [24, 112, 127], suggesting that simply extending the carrier sense mechanism to the receiver does not solve the problem. We argue that schemes like CSMA in which nodes use heuristics (such as "carrier is busy") to perform channel access are too conservative in exploiting concurrency because they are "proactive": nodes defer to ongoing transmissions without knowing whether in fact their transmission actually interferes with ongoing transmissions.

To improve throughput in a wireless network, we have proposed CMAP [123], a link layer whose channel access scheme *reactively* and *empirically* learns of transmission conflicts in the network. Nodes optimistically assume that concurrent transmissions will succeed, and carry them out in parallel. Then, in response to observed packet loss, they use the SoftPHY interface to discover which concurrent transmissions are likely to work, and which are likely to corrupt each other. This gives rise to a distributed data structure containing a "map" of conflicting transmissions (e.g., **S** to **R** and **CS** to **CR** in Figure 6-1). Nodes maintain the map in a distributed fashion by overhearing ongoing transmissions and exchanging lightweight information

with their one-hop neighbors. Then, by listening to ongoing transmissions on the shared medium to identify the current set of transmitters, and consulting the conflict map just before it intends to transmit, each node determines whether to transmit data immediately, or defer.

Of course, not all conflicting senders are in range of each other to overhear and make the transmit-or-defer decision because of the well-known "hidden terminal" problem (**HS** in Figure 6-1). To prevent performance degradation in such cases, a CMAP sender implements a reactive *loss-based backoff mechanism* to reduce its packet transmission rate in response to receiver feedback about packet loss. Finally, note that any scheme that seeks to exploit the exposed terminal opportunity shown in Figure 6-1 must cope with link-layer ACKs from **R** to **S** being lost at **S** due to a collision with **ES**'s transmission. CMAP tolerates ACK losses with a *windowed ACK and retransmission protocol.*

In recent work [123], we present CMAP's performance improvements in a large 802.11a wireless network testbed using commodity hardware to approximate the functionality provided by the SoftPHY interface.

6.1.2 Forwarding in a mesh network

SoftPHY has the capacity to improve the performance of mesh network routing-layer protocols such as opportunistic routing [12] and network coding. Using PPR, nodes need only forward or combine the bits likely to be correct in a packet that does not pass checksum, thus improving network capacity. Rather than use PP-ARQ, the integrated MAC/link layer that implements ExOR or network coding can directly work with SoftPHY's output. Alternatively, PP-ARQ could operate in the "background" recovering erroneous data, while the routing protocol sends the correct bits forward. Katti et al. take the former approach [64], integrating network coding with an opportunistic routing protocol that forwards only bits that SoftPHY labels as "good."

6.1.3 Multi-radio diversity

SoftPHY can improve the performance of multi-radio diversity (MRD) schemes [81] in which multiple access points listen to a transmission and combine the data to recover errors before forwarding the result, saving on retransmissions. Avudainayagan [4, 130] et al. develop a scheme in which multiple nodes (e.g., access points) exchange soft decision estimates of each data symbol and collaboratively use that information to improve decoding

performance. For this application, SoftPHY hints provide a way to design a protocol that does not rely on the specifics of the physical layer, unlike this previous work. Thus, with SoftPHY, we may be able to obtain the simpler design and PHY-independence of the block-based combining of MRD [81], while also achieving the performance gains associated with using physical-layer information. Woo et al. [131] propose a system for combining symbols from multiple receptions of a packet at different wireless access points using maximal ratio combining [97] of the soft information of each symbol.

6.2 FINAL THOUGHTS

In this dissertation, we started from an observation that in many wireless systems, bits in errored packets do not share fate. We then progressed to describe the design, implementation, and experimental evaluation of systems that use the SoftPHY interface (Chapter 3) and postamble-based decoding (Chapter 4). The SoftPHY interface is a small (in implementation complexity) modification to the physical layer to compute confidence information about each group of bits passed up to higher layers. Postamble-based decoding scheme recovers bits even when a packet's preamble has been corrupted. The SoftPHY interface and postamble-based decoding help higher layers perform better, as shown in Chapters 5 and the current chapter. Chapter 5 introduced our first application of the SoftPHY interface, PP-ARQ, which shows how a receiver can use this information together with a dynamic programming algorithm to efficiently request the sender to re-send small parts of packets, rather than an entire packet. Finally, in this chapter, we have surveyed a variety of protocols that benefit from the additional information that the SoftPHY interface provides.

We have implemented all of the designs from Chapters 1–5 in three different radio systems: IEEE 802.15.4 ("Zigbee" low-power wireless), IEEE 802.11a/g ("WiFi" OFDM local-area networking), and an uncoded DQPSK system. Our implementations are described in the above-referenced chapters.

We have evaluated components on the GNU Radio platform for 802.15.4, the Zigbee standard, and evaluated the components and system in a 31-node wireless testbed. Our results show a 2.1× improvement in throughput over the status quo under moderate load.

We believe that SoftPHY has the potential to change the way physical layer, link, and network protocol designers think about protocols. Today, wireless physical layer implementations employ significant amounts of redundancy to tolerate worst-case channel conditions. If noise during one or

more codewords is higher than expected, existing physical layers will generate incorrect bits, which will cause packet-level checksums to fail and require retransmission of the whole packet. Since interference fluctuations are often large, and the penalty for incorrect decoding is also large, physical layers tend to conservatively include lots of redundancy in the form of a high degree of channel coding or conservative modulation. Similarly, MAC layers tend to be quite conservative with rate adaptation because the consequences of errors are considered dire. The mind set seems to be that bit errors are undesirable and must be reduced to a very low rate (though eliminating them is impossible). As a result, they operate with comparatively low payload bit-rates.

The SoftPHY interface reduces the penalty of incorrect decoding, and thus for a given environment allows the amount of redundancy to be decreased, or equivalently the payload bit-rate to be increased. Put another way, with SoftPHY it may be perfectly fine for a physical layer to design for one or even two orders-of-magnitude higher BER, because higher layers need no longer cope with high packet error rates, but can decode and recover partial packets correctly.

Bibliography

[1] N. Abramson. The ALOHA system–Another Alternative for Computer Communications. In *Proc. of the Fall Joint Computer Conf., AFIPS Conf.*, volume 37, pages 281–285, 1970.

[2] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-Level Measurements From an 802.11b Mesh Network. In *Proc. of the ACM SIGCOMM Conf.*, pages 121–132, Portland, OR, Aug. 2004.

[3] J.-S. Ahn, S.-W. Hong, and J. Heidemann. An Adaptive FEC Code Control Algorithm for Mobile Wireless Sensor Networks. *Journal of Communications and Networks*, 7(4):489–499, 2005.

[4] A. Avudainayagam, J. M. Shea, T. F. Wong, and L. Xin. Reliability Exchange Schemes for Iterative Packet Combining in Distributed Arrays. In *Proc. of the IEEE WCNC Conf.*, volume 2, pages 832–837, Mar. 2003.

[5] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate. *IEEE Trans. on Information Theory*, 20(2):284–287, Mar. 1974.

[6] D. Barry, E. Lee, and D. Messerschmitt. *Digital Communication*. Springer, New York, NY, 3rd. edition, 2003.

[7] J. R. Barry, A. Kavčić, S. W. McLaughlin, A. Nayak, and W. Zhang. Iterative Timing Recovery. *IEEE Signal Processing Magazine*, 21(1):89–102, Jan. 2004.

[8] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes. In *Proc. of the IEEE ICC Conf.*, pages 54–83, Geneva, Switzerland, May 1993.

[9] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, 2nd. edition, 1987.

[10] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: A Media-Access Protocol for Packet Radio. In *Proc. of the ACM SIGCOMM Conf.*, pages 212–225, London, England, Aug. 1994.

[11] J. Bicket. Bit-Rate Selection in Wireless Networks. Master's thesis, Massachusetts Institute of Technology, Feb. 2005.

[12] S. Biswas and R. Morris. ExOR: Opportunistic Multi-hop Routing for Wireless Networks. In *Proc. of the ACM SIGCOMM Conf.*, pages 133–144, Philadelphia, PA, Aug. 2005.

[13] F. Calì, M. Conti, and E. Gregori. Dynamic Tuning of the IEEE 802.11 Protocol to Achieve a Theoretical Performance Limit. *IEEE/ACM Trans. on Networking*, 8(6):785–799, December 2000.

[14] M. Cesana, D. Maniezzo, P. Bergamo, and M. Gerla. Interference Aware (IA) MAC: an Enhancement to IEEE 802.11b DCF. In *Proc. of the IEEE Vehicular Technology Conf.*, volume 5, pages 2799–2803, Oct. 2003.

[15] D. Chase. Code Combining: A Maximum-Likelihood Decoding Approach for Combining an Arbitrary Number of Noisy Packets. *IEEE Trans. on Comm.*, 33(5):385–393, May 1985.

[16] I. Chlamtac and A. Faragó. Making Transmission Schedules Immune to Topology Changes in Multihop Packet Radio Networks. *IEEE/ACM Trans. on Networking*, 2(1):23–29, April 1994.

[17] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[18] D. Costello, Jr., J. Hagenauer, H. Imai, and S. B. Wicker. Applications of Error-Control Coding. *IEEE Trans. on Info. Theory*, 44(6):2531–2560, Oct. 1998.

[19] G. R. Danesfahani and T. G. Jeans. Optimisation of Modified Mueller and Müller Algorithm. *Electronics Letters*, 31(13):1032–1033, June 1995.

[20] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proc. of the ACM MobiCom Conf.*, pages 134–146, San Diego, Sept. 2003.

[21] S. Desilva and R. Boppana. On the Impact of Noise Sensitivity on Performance in 802.11 Based Ad Hoc Networks. In *Proc. of the IEEE Intl. Conf. on Communication*, Paris, France, June 2004.

[22] H. Dubois-Ferrière, D. Estrin, and M. Vetterli. Packet Combining in Sensor Networks. In *Proc. of the ACM SenSys Conf.*, pages 102–115, San Diego, CA, Nov. 2005.

[23] S. Eisenman and A. Campbell. Structuring Contention-Based Channel Access in Wireless Sensor Networks. In *Proc. of the ACM/IEEE IPSN Conf.*, pages 226–234, Nashville, TN, Apr. 2006.

[24] S. Eisenmann and A. Campbell. E-CSMA: Supporting Enhanced CSMA Performance in Experimental Sensor Networks using Per-neighbor Transmission Probability Thresholds. In *Proc. of the IEEE INFOCOM Conf.*, pages 1208–1216, Anchorage, AK, May 2007.

[25] M. Ettus. Ettus Research, LLC. See `http://www.ettus.com`.

[26] European Telecommunication Standard. High Performance Radio Local Area Network (HIPERLAN) Type 1; Functional Specification, 1996.

[27] G. D. Forney, Jr. The Viterbi Algorithm (Invited Paper). *Proc. of the IEEE*, 61(3):268–278, Mar. 1973.

[28] J. Fuemmeler, N. Vaidya, and V. Veeravalli. Selecting Transmit Powers and Carrier Sense Thresholds for CSMA Protocols. Technical report, University of Illinois at Urbana-Champaign, October 2004.

[29] C. Fullmer and J. J. Garcia-Luna-Aceves. FAMA-PJ: a Channel Access Protocol for Wireless LANs. In *Proc. of the ACM MOBICOM Conf.*, pages 76–85, Berkeley, CA, 1995.

[30] C. Fullmer and J. J. Garcia-Luna-Aceves. Floor Acquisition Multiple Access for Packet Radio Networks. In *Proc. of the ACM SIGCOMM Conf.*, pages 262–273, Cambridge, MA, 1995.

[31] C. Fullmer and J. J. Garcia-Luna-Aceves. Solutions to Hidden Terminal Problems in Wireless Networks. In *Proc. of the ACM SIGCOMM Conf.*, pages 39–49, Cannes, France, Sept. 1997.

[32] R. G. Gallager. A Perspective on Multiaccess Channels. *IEEE Trans. on Information Theory*, IT-31(2):124–142, Mar. 1985.

[33] J. A. Gansman, J. V. Krogmeier, and M. P. Fitz. Single Frequency Estimation with Non-Uniform Sampling. In *Proc. of the Asilomar Conf. on Signals, Systems, and Computers*, volume 1, pages 399–403, Pacific Grove, CA, November 1996.

[34] R. Ganti, P. Jayachandran, H. Luo, and T. Abdelzaher. Datalink Streaming in Wireless Sensor Networks. In *Proc. of the ACM SenSys Conf.*, pages 209–222, Boulder, CO, Nov. 2006.

[35] R. Garcés and J. J. Garcia-Luna-Aceves. Floor Acquisition Multiple Access with Collision Resolution. In *Proc. of the ACM MOBICOM Conf.*, pages 187–197, Rye, NY, 1996.

[36] R. Garcés and J. J. Garcia-Luna-Aceves. A Near-Optimum Channel Access Protocol Based on Incremental Collision Resolution and Distributed Transmission Queues. In *Proc. of the IEEE INFOCOM Conf.*, volume 1, pages 158–165, San Francisco, CA, Mar. 1998.

[37] F. Gardner. Interpolation in Digital Modems—Part I: Fundamentals. *IEEE Trans. on Comm.*, 3(41):501–507, March 1993.

[38] F. Gardner. *Phaselock Techniques*. Wiley-Interscience, Hoboken, NJ, 3rd. edition, 2005.

[39] The GNU Radio Project. `http://www.gnu.org/software/gnuradio`.

[40] S. Godtmann, A. Pollok, N. Hadaschik, G. Ascheid, and H. Meyr. On the Influence of Pilot Symbol and Data Symbol Positioning on Turbo Synchronization. In *Proc. of the IEEE VTC Conf.*, pages 1723–1726, April 2007.

[41] A. Goldsmith. *Wireless Communications*. Cambridge University Press, Cambridge, U.K., 2005.

[42] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Trans. on Information Theory*, 46(2):388–404, Mar. 2000.

[43] J. Hagenauer. Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and their Applications. *IEEE Trans. on Comm.*, 36(4):389–400, Apr. 1988.

[44] J. Hagenauer and P. Hoeher. A Viterbi Algorithm with Soft-Decision Outputs and its Applications. In *Proc. of the IEEE GLOBECOM Conf.*, pages 1680–1686, Dallas, TX, Nov. 1989.

[45] Harris Semiconductor HSP3824 Direct Sequence Spread Spectrum Baseband Processor Datasheet, Mar. 1996. http://www.chipdocs.com/pndecoder/datasheets/HARIS/HSP3824.html.

[46] C. Herzet, N. Noels, V. Lottici, H. Wymeersch, M. Luise, M. Moeneclaey, and L. Vandendorpe. Coded-Aided Turbo Synchronization (Invited Paper). *Proc. of the IEEE*, 95(6):1255–1271, June 2007.

[47] High Speed Packet Access vendor web site. See http://hspa.gsmworld.com.

[48] G. Holland, N. Vaidya, and P. Bahl. A Rate-Adaptive MAC Protocol for Multihop Wireless Networks. In *Proc. of the ACM MobiCom Conf.*, pages 236–251, Rome, Italy, Sept. 2001.

[49] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating Congestion in Wireless Sensor Networks. In *Proc. of the ACM SenSys Conf.*, pages 134–147, Baltimore, MD, November 2004.

[50] IEEE Standard 802.11-1999: Wireless LAN Medium Access Control and Physical Layer Specifications, August 1999. http://standards.ieee.org/getieee802/802.11.html.

[51] IEEE Standard 802.11b-1999: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band, Sept. 1999. http://standards.ieee.org/getieee802/802.11.html.

[52] IEEE Standard 802.11a-2003: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 5 GHz Band, June 2003. http://standards.ieee.org/getieee802/802.11.html.

[53] IEEE Standard 802.11g-2003: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 4: Further Higher Data Rate Extension in the 2.4 GHz Band, June 2003. http://standards.ieee.org/getieee802/802.11.html.

[54] IEEE Standard 802.15.4-2003: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), Oct. 2003. http://standards.ieee.org/getieee802/802.15.html.

[55] IEEE Standard 802.16-2004: Air Interface for Fixed Broadband Wireless Access Systems, Oct. 2004. `http://standards.ieee.org/getieee802/802.16.html`.

[56] IEEE Standard 802.16e-2005: Air Interface for Fixed and Mobile Broadband Wireless Access Systems, Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands, Feb. 2006. `http://standards.ieee.org/getieee802/802.16.html`.

[57] J Padhye and S Agarwal and V Padmanabhan and L Qiu and A Rao and B Zill. Estimation of Link Interference in Static Multi-hop Wireless Networks. In *ACM/USENIX Internet Measurement Conference*, Berkeley, CA, Oct. 2005.

[58] K. Jamieson and H. Balakrishnan. PPR: Partial Packet Recovery for Wireless Networks. In *Proc. of the ACM SIGCOMM Conf.*, pages 409–420, Kyoto, Japan, Aug. 2007.

[59] K. Jamieson, B. Hull, A. Miu, and H. Balakrishnan. Understanding the Real-World Performance of Carrier Sense. In *Proc. of the ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design*, pages 52–57, Philadelphia, PA, Aug. 2005.

[60] Y. Jeong and J. Lehnert. Acquisition of Packets with a Short Preamble for Direct-Sequence Spread-Spectrum Multiple-Access Packet Communications. In *Proc. of IEEE MILCOM Conf.*, pages 1060–1064, Boston, MA, 2003.

[61] G. Judd. *Using Physical Layer Emulation to Understand and Improve Wireless networks.* PhD thesis, CMU, October 2006. CMU-CS-06-164.

[62] A. Kamerman and L. Monteban. WaveLAN II: a High-Performance Wireless LAN for the Unlicensed Band. *Bell Labs Technical Journal*, 2(3):118–133, Summer 1997.

[63] P. Karn. MACA—A New Channel Access Method for Packet Radio. In *Proc. of the 9th ARRL Computer Networking Conf.*, London, Ontario, 1990. See also: `http://www.ka9q.net/papers/maca.html`.

[64] S. Katti, H. Balakrishnan, and D. Katabi. Symbol-level Network Coding for Mesh Networks. In *Proc. of the ACM SIGCOMM Conf.*, Seattle, WA, 2008.

[65] S. M. Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory, Vol. I.* Prentice Hall, PTR, Upper Saddle River, NJ, 1st. edition, 1993.

[66] L. Kleinrock and F. Tobagi. Packet Switching in Radio Channels: Part I—Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics. *IEEE Trans. on Communications*, COM-23(12), Dec. 1975.

[67] R. M. Koteng. *Evaluation of SDR-implementation of IEEE 802.15.4 Physical Layer*. PhD thesis, Norwegian University of Science and Technology, July 2006.

[68] L. E. Larson, P. Asbeck, and D. Kimball. Multifunctional RF Transmitters for Next Generation Wireless Transceivers. In *Proc. of the IEEE Intl. Symp. on Circuits and Systems*, pages 753–756, May 2007.

[69] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee, and R. Morris. Capacity of Ad Hoc Wireless Networks. In *Proc. of the ACM MobiCom Conf.*, pages 61–69, Rome, Italy, July 2001.

[70] S. Lin and D. J. Costello. *Error Control Coding*. Prentice Hall, Upper Saddle River, NJ, 2nd. edition, 2004.

[71] S. Lin and P. S. Yu. A Hybrid ARQ Scheme with Parity Retransmission for Error Control of Satellite Channels. *IEEE Trans. on Comm.*, 30(7):1701–1719, July 1982.

[72] Multiband Atheros Driver for Wireless Fidelity. See `http://madwifi.org`.

[73] D. Mandelbaum. An Adaptive-Feedback Coding Scheme Using Incremental Redundancy (Corresp.). *IEEE Trans. on Information Theory*, 20(3):388–389, May 1974.

[74] R. Mehlan, Y.-E. Chen, and H. Meyr. A Fully Digital Feedforward MSK Demodulator with Joint Frequency Offset and Symbol Timing Estimation for Burst Mode Mobile Radio. *IEEE Trans. on Vehicular Technology*, 42(4):434–443, November 1993.

[75] U. Mengali and A. N. D'Andrea. *Synchronization Techniques for Digital Receivers*. Kluwer Academic/Plenum Publishers, 1st. edition, Oct. 1997.

[76] R. M. Metcalfe and D. R. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks. *Communications of the ACM*, 19(7):395–404, July 1976.

[77] R. M. Metcalfe, D. R. Boggs, C. P. Thacker, and B. W. Lampson. U.S. Patent No. 4,063,220: Multipoint Data Communication System with Collision Detection, Dec. 1977.

[78] J. Metzner. Improvements in Block-Retransmission Schemes. *IEEE Trans. on Comm.*, 27(2):524–532, Feb. 1979.

[79] H. Meyr, M. Moeneclaey, and S. A. Fechtel. *Digital Communication Receivers: Synchronization, Channel Estimation, and Signal Processing*. John Wiley & Sons, New York, NY, 1998.

[80] J. Mitola. The Software Radio Architecture. *IEEE Communications Magazine*, 33(5):26–38, May 1995.

[81] A. Miu, H. Balakrishnan, and C. E. Koksal. Improving Loss Resilience with Multi-Radio Diversity in Wireless Networks. In *Proc. of the ACM MobiCom Conf.*, pages 16–30, Cologne, Germany, Aug. 2005.

[82] O. Mowafi and A. Ephremides. Analysis of a Hybrid Access Scheme for Buffered Users—Probabilistic Time Division. *IEEE Trans. on Software Engineering*, 8(1):52–60, January 1982.

[83] K. Mueller and M. Müller. Timing Recovery in Digital Synchronous Data Receivers. *IEEE Trans. on Comm.*, 24(5), May 1976.

[84] N. Noels, H. Steendam, M. Moeneclaey, and H. Bruneel. A Maximum Likelihood Based Feedback Carrier Synchronizer for Turbo-Coded Systems. In *Proc. of the IEEE VTC Conf.*, volume 5, pages 3202–3206, May 2005.

[85] N. Noels, H. Steendam, M. Moeneclaey, and H. Bruneel. Carrier Phase and Frequency Estimation for Pilot-Symbol Assisted Transmission: Bounds and Algorithms. *IEEE Transactions on Signal Processing*, 53(12):4578–4587, Dec. 2005.

[86] ONOE Rate Control. See http://madwifi.org/browser/trunk/ath_rate/onoe.

[87] A. V. Oppenheim and G. C. Verghese. Signals, Systems, and Inference: Class Notes for 6.011 Introduction to Communications, Control, and Signal Processing, 2006.

[88] A. V. Oppenheim and A. S. Wilsky. *Signals and Systems*. Prentice-Hall, Upper Saddle River, NJ, 2nd. edition, 1996.

[89] S. Pasupathy. Minimum Shift Keying: A Spectrally-Efficient Modulation. *IEEE Communications Magazine*, 7(4):14–22, July 1979.

[90] B. Pearson. Complementary Code Keying Made Simple, May 2000. Intersil Application Note AN9850.1.

[91] M. H. Perrott. The CppSim Behavioral Simulator. See http://www-mtl.mit.edu/researchgroups/perrottgroup/tools.html.

[92] M. H. Perrott. *Techniques for High Data Rate Modulation and Low Power Operation of Fractional-N Frequency Synthesizers*. PhD thesis, Massachusetts Institute of Technology, Sept. 1997.

[93] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach*. Morgan-Kaufmann, 3rd. edition, 2003.

[94] J. Polastre, J. Hill, and D. Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proc. of the ACM SenSys Conf.*, pages 95–107, Baltimore, MD, November 2004.

[95] A. Polydoros and C. Weber. A Unified Approach to Serial Search Spread-Spectrum Code Acquisition—Part I: General Theory. *IEEE Trans. on Communications*, 32(5):542–549, May 1984.

[96] N. B. Priyantha. *The Cricket Indoor Location System*. PhD thesis, MIT, May 2005.

[97] J. G. Proakis. *Digital Communications*. McGraw-Hill, New York, NY, 4th. edition, 2001.

[98] E. Purcell. *Electricity and Magnetism*. McGraw-Hill, 2nd. edition, 1985.

[99] R. Rao et al. CalRadio I. See http://calradio.calit2.net/calradio1a.htm.

[100] T. S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall PTR, Upper Saddle River, NJ, 2002.

[101] I. Rhee, A. Warrier, M. Aia, and J. Min. Z-MAC: A Hybrid MAC for Wireless Sensor Networks. In *Proc. of the ACM SenSys Conf.*, pages 90–101, San Diego, CA, November 2005.

[102] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly. Opportunistic Media Access for Multirate Ad Hoc Networks. In *Proc. of the ACM MobiCom Conf.*, pages 24–35, Atlanta, GA, Sept. 2002.

[103] J. H. Saltzer. End-to-end Arguments in System Design. *ACM Transmisions on Computer Systems*, 2(4):277–288, Nov. 1984.

[104] J. H. Saltzer. Personal comunication, Apr. 2008.

[105] T. K. Sarkar et al. *History of Wireless*. Wiley-Interscience, Hoboken, NJ, 2006.

[106] T. Schmid. Personal communication, Sept. 2006.

[107] T. Schmid et al. GNURadio UCLA SVN repository. See http://acert.ir.bbn.com/svn/?group=gr-ucla.

[108] T. M. Schmidl and D. C. Cox. Robust Frequency and Timing Synchronization for OFDM. *IEEE Trans. on Comm.*, 45(12):1613–1621, Dec. 1997.

[109] T. J. Shepard. *Decentralized Channel Management in Scalable Multihop Spread-Spectrum Packet Radio Networks*. PhD thesis, Massachusetts Institute of Technology, July 1995.

[110] R. Sinha, C. Papadopoulos, and J. Heidemann. Internet Packet Size Distributions: Some Observations. See http://netweb.usc.edu/~rsinha/pkt-sizes.

[111] E. Soljanin, N. Varnica, and P. Whiting. Punctured vs Rateless Codes for Hybrid ARQ. In *Proc. of the IEEE Information Theory Workshop*, pages 155–159, Punta del Este, Uruguay, Mar. 2006.

[112] D. Son, B. Krishnamachari, and J. Heidemann. Experimental Analysis of Concurrent Packet Transmissions in Low-Power Wireless Networks. In *Proc. of the SenSys Conf.*, pages 237–250, Boulder, CO, Nov. 2006.

[113] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. Understanding the Causes of Packet Delivery Success and Failure in Dense Wireless Sensor Networks. Technical Report SING-06-00, Stanford Univ., 2006. `http://sing.stanford.edu/pubs/sing-06-00.pdf`.

[114] K. Srinivasan and P. Levis. RSSI is Under Appreciated. In *Proc. of the EmNets Workshop*, 2006.

[115] C. P. Thacker. Personal Distributed Computing: the Alto and Ethernet Hardware. In *Proc. of the ACM Conf. on the History of Personal Workstations*, pages 87–100, Jan. 1986.

[116] TI/Chipcon Corp. CC1000 Transceiver Datasheet. See `http://focus.ti.com/lit/ds/symlink/cc1000.pdf`.

[117] TI/Chipcon Corp. CC2420 Transceiver Datasheet. See `http://www.ti.com/lit/gpn/cc2420`.

[118] F. Tobagi. Multiaccess Protocols in Packet Communication Systems. *IEEE Trans. on Communications*, 28(4):468–488, April 1980.

[119] F. Tobagi and L. Kleinrock. Packet Switching in Radio Channels: Part III—Polling and (Dynamic) Split-Channel Reservation Multiple Access. *IEEE Trans. on Communications*, COM-24(8), Aug. 1976.

[120] D. Tse and P. Viswanath. *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.

[121] H. L. Van Trees. *Detection, Estimation, and Modulation Theory: Part I*. John Wiley & Sons, Hoboken, NJ, paperback edition, 2001.

[122] S. Verdú. Minimum Probability of Error for Asynchronous Gaussian Multiple-access Channels. *IEEE Trans. on Information Theory*, IT-32(1):85–97, Jan. 1986.

[123] M. Vutukuru, K. Jamieson, and H. Balakrishnan. Harnessing Exposed Terminals in Wireless Networks. In *Proc. of the USENIX NSDI Conf.*, pages 59–72, San Francisco, CA, Apr. 2008.

[124] C.-Y. Wan, S. Eisenman, and A. Campbell. CODA: Congestion Detection and Avoidance in Sensor Networks. In *Proc. of the ACM SenSys Conf.*, pages 266–279, Los Angeles, CA, November 2003.

[125] X. Wang and H. V. Poor. Iterative (Turbo) Soft Interference Cancellation and Decoding for Coded CDMA. *IEEE Trans. on Communications*, 47(7):1046–1061, July 1999.

[126] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: A Wireless Sensor Network Testbed. In *Proc. of the IPSN Conf., Special Track on Platform Tools and Design Methods for Network Embedded Sensors (SPOTS)*, Apr. 2005. See also `http://motelab.eecs.harvard.edu`.

[127] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the Capture Effect for Collision Detection and Recovery. In *IEEE EmNets Workshop*, Sydney, Australia, May 2005.

[128] J. K. Wolf. Efficient Maximum-Likelihood Decoding of Linear Block Codes Using a Trellis. *IEEE Trans. Information Theory*, IT-24(1):76–80, Jan. 1978.

[129] S. Wong, H. Yang, S. Lu, and V. Bharghavan. Robust Rate Adaptation for 802.11 Wireless Networks. In *Proc. of ACM MobiCom*, pages 146–157, Los Angeles, CA, Sept. 2006.

[130] T. F. Wong, L. Xin, and J. M. Shea. Iterative Decoding in a Two-Node Distributed Array. In *Proc. of the IEEE MILCOM Conf.*, volume 2, pages 1320–1324, Oct. 2002.

[131] G. Woo, P. Keradpour, D. Shen, and D. Katabi. Beyond the Bits: Cooperative Packet Recovery Using Physical Layer Information. In *Proc. of the ACM MobiCom Conf.*, pages 147–158, Montreal, Quebec, Canada, September 2007.

[132] X. Yang and N. Vaidya. On Physical Carrier Sensing in Wireless Ad Hoc Networks. In *Proc. of the IEEE INFOCOM Conf.*, volume 4, pages 2525–2535, Miami, FL, Mar. 2005.

[133] W. Ye, J. Heidemann, and D. Estrin. An Energy-efficient MAC Protocol for Wireless Sensor Networks. In *Proc. of the IEEE INFOCOM Conf.*, volume 3, pages 1567–1576, June 2002.

[134] J. Zhu, X. Guo, L. L. Yang, W. S. Conner, S. Roy, and M. M. Hazra. Adapting Physical Carrier Sensing to Maximize Spatial Reuse in 802.11 Mesh Networks. *Wiley Journal of Wireless Communications and Mobile Computing*, 4(8):933–946, December 2004.