Physical Layer-Aware Wireless Link Layer Protocols

by

Mythili Ranganath Vutukuru

S.M., Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2006)

B. Tech., Computer Science and Engineering, Indian Institute of Technology, Madras (2004)

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 2010

Author . . . . . .
Department of Electrical Engineering and Computer Science
May 21, 2010

Certified by . . . . . .
Hari Balakrishnan
Professor
Thesis Supervisor

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

Physical Layer-Aware Wireless Link Layer Protocols

by

Mythili Ranganath Vutukuru

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

With wireless devices becoming ubiquitous, the problem of designing high performance and reliable wireless networks is of great importance today. Wireless links are characterized by a rapidly varying channel, requiring transmitters to dynamically adapt their transmit bit rate. The broadcast nature of radio also necessitates the use of medium access protocols to arbitrate access among competing transmitters and reduce losses due to interference, while enabling successful concurrent transmissions. We observe that the problems with existing bit rate adaptation and medium access protocols stem from insufficient information about the wireless channel at the link layer. This dissertation makes two contributions: (i) a redesign of the interface between the physical and link layers in wireless networks to expose more information about the wireless channel to the link layer, and (ii) the design and evaluation of new link-layer protocols that improve throughput by using information about the channel delivered via the new interface.

In today's network architecture, the physical layer (PHY) delivers received frames and per-frame signal strength measurements to the link layer. This dissertation proposes two enhancements to this interface: the PHY *streams* bits to the link layer as soon as they are decoded and before the entire frame reception completes, and it computes and exports *SoftPHY hints* with each decoded bit. The SoftPHY hint of a bit indicates the PHY's confidence that the decoded bit is correct. We show that the SoftPHY hints of a received frame can be used to estimate the bit error rate (BER) of the wireless channel faster and with more accuracy than with existing methods. We develop the *SoftRate* bit rate adaptation protocol that uses this BER computed from SoftPHY hints to pick transmit bit rates and improves throughput by $2\times$ over existing protocols. The streaming PHY interface enables the link layer to learn about the current transmission on the air by decoding headers before frame reception completes. The *SoftCMAP* and *CMAP* protocols make smart medium access decisions using this knowledge of ongoing transmissions along with a distributed map of conflicting transmissions, and improve aggregate network throughput by up to 50% by increasing the number of successful concurrent transmissions.

Thesis Supervisor: Hari Balakrishnan
Title: Professor

To Mom and Dad.

Acknowledgments

Grad school has been the most enriching period of my life so far, both personally and professionally, thanks to the many wonderful people I have had the good fortune of meeting and learning from in the past six years. The list begins with my advisor, Hari Balakrishnan. Hari's passion for his work, his clarity of thought, his knack of coming up with creative solutions to the problem at hand, and most of all, his ability to accomplish so much and yet remain completely relaxed are truly inspiring. Hari trusted me to do things I never knew I was capable of, and gave me complete freedom to work and figure things out at my own pace. And besides being a great advisor, he has also been a caring mentor and guide on matters beyond research. Thank you, Hari.

I thank Robert Morris and Dina Katabi for serving on my thesis committee. Their insightful comments have greatly improved this dissertation. I thank Prof. Arvind, Alfred Ng, Kermin Elliott Fleming, and the rest of the Airblue team for helping with some of the implementation work in this dissertation, and for patiently educating me on various hardware issues. I am grateful to Kyle Jamieson, a collaborator on the work in this dissertation, for teaching me the many subtleties of wireless systems over the past few years. Thanks also to Vern Paxson for the very enjoyable summer I spent at the International Computer Science Institute at Berkeley in 2006.

I have benefited immensely from the wisdom and expertise of my officemates and colleagues during my stay at CSAIL. Mike Walfish taught me a great deal by example, and was a big influence in my formative years as a grad student. I am grateful to him for valuable lessons on the science of conducting thorough experimental evaluations, and on the art of articulating one's work clearly. My officemates in 32-G982 over the years—Dave Andersen, Jakob Eriksson, Nick Feamster, Jaeyeon Jung, Ramakrishna Gummadi, Katrina LaCurts, Raluca Ada Popa, Emil Sit, Lenin Sivalingam, and Arvind Thiagarajan—provided great company, helpful suggestions, and many intellectually stimulating discussions. I would also like to thank the current and former students of NMS—Magdalena Balazinska, Vladimir Bychkovsky, Shyamnath Gollakota, Rahul Hariharan, Bret Hull, Szymon Jakubczak, Srikanth Kandula, Sachin Katti, Nate Kushman, Allen Miu, Samuel David Perli, Stan Rost, and others—for their friendship, and for critical comments on paper drafts and practice talks. I am grateful to Michel Goraczko and Dorothy Curtis for always being willing to help with various systems issues. Thanks also to Sheila Marian for taking care of the administrative issues, and for baking yummy goodies.

I will forever cherish the friends I have made over the years at MIT. Jayashree was a great companion, both in the lab and outside of it. I will fondly remember our many long walks along the river every summer. I am glad to have had Lavanya as my roommate all these years to share my grad school experience with. I thank Krishna, Prabha, Arvind, Vivek, and Jaykumar for their delicious home-cooked dinners and accompanying conversations. Thanks also to Viji, Nishanth, Madhu, Saumya, and Vidya for their friendship. Finally, thanks to all the friends I have made through MIT Samskritam for making the experience of learning a new language so much fun.

Going back in time, I am grateful to my teachers in school and undergrad for introducing me to the joy of learning. I would also like to thank my best friends for the past ten years, Madhuri and Kanu, for being the elder sisters I never had growing up.

My family always makes me feel like I am the luckiest girl on earth. My parents-in-law and sister-in-law have been a great source of support the past few months, and I am glad to have become a part of the Tatavarthy household. My uncles, aunts, and cousins in Detroit and San Jose have provided me a home away from home in this country. Bamma, ammamma, Ravi babai, Padmaja pinni, and the rest of my extended family have showered me with love and affection right from my childhood. And my "chelli" Sravanthi is the cutest little sister one can ask for. I know words aren't enough, but thank you all!

My parents, Phani Rajakumari and Ranganath, have loved and supported me unconditionally all my life. I consider myself very fortunate for being their daughter, and I dedicate this dissertation to them. My husband Vamsi is one of the most level-headed persons I have ever met, and has been the source of my sanity through the lows in grad school. Turning in this dissertation also marks the end of our long-distance relationship, and I look forward to beginning a new chapter of our life together.

Contents

List of Figures

List of Tables

Previously Published Material

Chapters 3 and 4 significantly revise a previous publication [67]: Mythili Vutukuru, Hari Balakrishnan, and Kyle Jamieson. Cross-Layer Wireless Bit Rate Adaptation. In *Proceedings of ACM SIGCOMM*, August, 2009.

Chapter 5 significantly revises a previous publication [68]: Mythili Vutukuru, Kyle Jamieson, and Hari Balakrishnan. Harnessing Exposed Terminals in Wireless Networks. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, April, 2008.

Chapter 1

Introduction

The technological improvements in the field of wireless communication have made wireless devices ubiquitous, from handheld computers and laptops with internet access to phones and other devices with cellular data connectivity. Although the rated wireless link speeds have been increasing, the end-to-end throughputs in wireless networks are often much lower than the link speeds, and fall far short of their wired counterparts.

This dissertation presents solutions that improve the throughput of wireless data networks. The challenges in wireless networks stem from the fundamental properties of radio and are markedly different from those in wired networks. The electrical circuitry in the wired channel adds a small amount of noise to transmissions, barring which the quality of the channel stays constant and predictable. For instance, a well-engineered 1 Gbps optical fiber will be able to consistently deliver a throughput close to 1 Gbps across the two ends of the fiber for a very long time. In contrast, the quality of the wireless channel changes rapidly with time. Radio is also a shared broadcast medium, unlike most wired channels that provide an exclusive channel of communication between the two end-points. Even in cases when the wire is shared (e.g., Ethernet), sharing is usually simpler because, unlike in wireless channels, all the devices on the cable can hear everyone else and can coordinate to transmit on the wire one at a time.

The wireless channel is more complicated than a wired channel for many reasons. Mobility of the transmitter or the receiver or scattering from the various objects in the environment changes the average signal strength between the sender and receiver. In addition, *multipath fading* that occurs due to the combining of multiple copies of the transmitted signal at the receiver leads to random fluctuations on shorter timescales. As a result, the strength of the transmitted signal at the receiver, as measured by the Signal to Noise Ratio (SNR), can change by many orders of magnitude within a short period of time. This change can result in drastic variations in the bit error rate (BER) of transmitted messages. For example, Figure 1-1 illustrates the variation of SNR and BER over time when a sender is moving away from the receiver at walking speed. Notice that the average signal quality gradually reduces from the left to right in the graph due to increasing distance between the sender and receiver, and multipath propagation causes random fluctuations over shorter timescales.

The wireless channel is a broadcast shared medium and the simultaneous transmissions of different senders interfere over the air. Now, some concurrent transmissions result in a

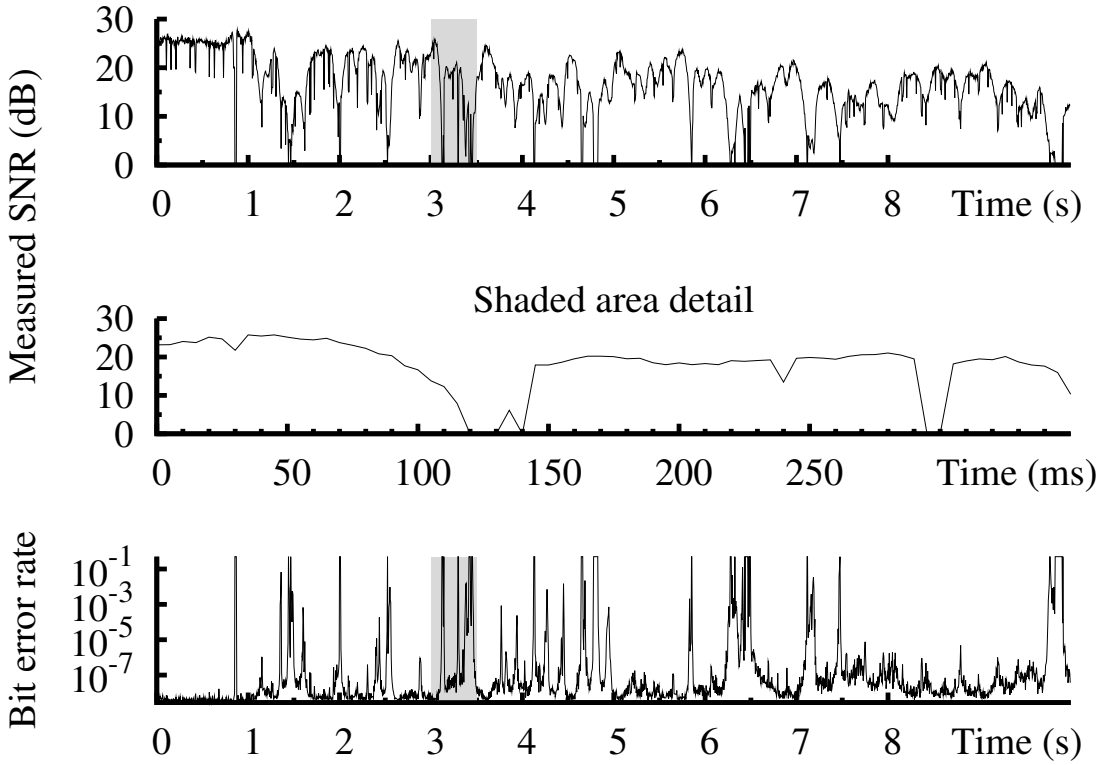


Figure 1-1: SNR fluctuations across time with pedestrian mobility. One can notice a gradual decline in the average signal strength (shown in the logarithmic dB scale) over the 10-second window in the topmost panel as the sender moves away from the receiver. If we zoom in on a 350 ms snapshot in the middle panel, we see variations due to multipath propagation that last a few tens of milliseconds. Changes in the signal strength also cause changes in the channel BER (measured at BPSK modulation and a code rate 1/2) by many orders of magnitude during this period, as shown in the bottom panel. Experimental data in this graph is obtained using an 802.11a/g-like software radio prototype (see Section 4.3).

garbled signal at some or all of the corresponding receivers, while some other concurrent transmissions can proceed successfully. Sharing the wireless medium therefore involves allowing as many concurrent transmissions as possible without causing unacceptable amount of interference to any of the transmissions. This problem is much harder than its analogue in wired networks because a receiver's ability to decode a packet successfully depends on channel conditions near the receiver, while the decision to transmit must be made by the sender. The receiver and sender are not always close enough that they experience similar noise and interference conditions. In Figure 1-2, when sender *S* is transmitting to *R*, it is best for the sender *CS* to not transmit to *CR*, because the interference from *S* will garble the reception at *CR*. However, sender *ES* can (or rather, it *should*, in order to maximize aggregate network throughput) transmit to *ER* when *S* is sending to *R* because destination *ER* is far enough from sender *S* and is unaffected by *S*. In fact, *ES* may be able to transmit successfully to *ER* even if *ER* is within hearing range of *S*—researchers have observed that

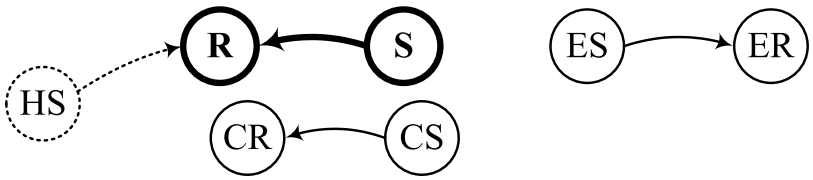


Figure 1-2: A sample transmission from *S* to *R* with three abstract sender cases: a conflicting sender *CS* that suffers interference if it transmits concurrently with *S*, an exposed sender *ES* that can successfully transmit concurrently to *ER* but is prevented from doing so by the CSMA MAC protocol, and a hidden sender *HS* that may not carrier sense sender *S* and may end up transmitting concurrently with *S* resulting in a loss.

receivers can often "capture" packets from a transmission even in the presence of other weak interfering transmissions [60, 70]. Therefore, the decision of when it is safe for a node to transmit depends on the other transmitters and the locations of their intended destinations, unlike in wired networks.

Most of the complexity of the wireless channel is managed by the physical (PHY) and data link layers of the wireless networking stack. The PHY receives frames from the link layer and transmits the bits over the wireless channel from the transmitter to the specified receiver using the RF front-end. Among other things, the PHY applies an error-correcting code to recover bit errors, and modulates the digital data into a form that can be transmitted as analog waveforms. Depending on the amount of redundancy added as part of the error-correcting code and the information-packing efficiency of the modulation scheme used, the PHY can transmit digital information at various *bit rates*. Transmissions at higher bit rates have lower redundancy and are more vulnerable to channel noise than transmissions at lower bit rates.

The PHY by itself does not handle the problems of channel variability and sharing; these problems are handled by various link-layer protocols. When the link layer hands a message to the PHY for transmission, it specifies the bit rate at which the message should be sent. This transmit bit rate is computed by the link layer using a *bit rate adaptation* protocol. By selecting bit rates that work best at the current channel quality, bit rate adaptation keeps the BER of transmitted messages within acceptable limits, hides packet losses due to channel variability from higher layers, and helps maximize link throughput. The link layer also decides when a frame should be transmitted using a *medium access control (MAC)* or *channel access* protocol. MAC protocols arbitrate fair access to the medium while trying to maximize aggregate network throughput by increasing spatial reuse. In today's layered network architecture, the link layer uses only a small amount of information about the state of the channel obtained via the PHY interface to make these rate adaptation and channel access decisions:

1. The link layer knows whether a transmitted frame was received correctly based on whether it elicited a link-layer ACK.

2. The PHY passes up an indicator of the sender's signal quality at the receiver, like SNR, along with every decoded frame.

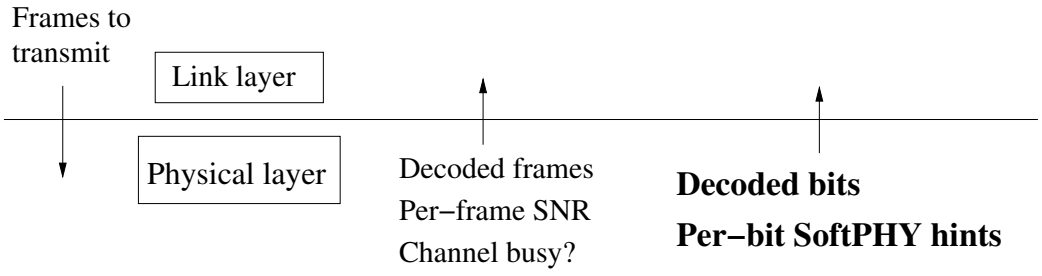


Figure 1-3: The interface between the physical and link layers in today's wireless network stack, and our proposed modifications (shown in bold).

3. The link layer can also poll the PHY to learn if the channel is "busy" with another transmission at a certain time by, say, having the PHY check if the energy of the received signal is above a certain threshold.

Despite being an active area of research for a long time, existing bit rate adaptation and MAC protocols still leave room for improvement. This dissertation argues that the common source of sub-optimality across various link-layer protocols is the lack of sufficient information to make intelligent decisions, which is a result of the rather narrow PHY interface described above. The PHY has a great deal of useful information about the state of the channel that is not exposed to the link layer, in part because the current link layer-PHY interface was designed with wired networks in mind. For example, current rate adaptation protocols use as channel quality feedback either frame reception events (which provide only a very small amount of information per frame about channel quality) or per-frame SNR measurements (which must be mapped to channel BER or link-layer throughput using either theoretical models of the propagation channel or empirical measurements). However, the PHY often has more detailed information on how good or bad the reception of individual bits in a packet was, which can be used to precisely estimate the channel bit error rate and pick suitable bit rates. The PHY also has useful information for MAC protocols. Today's MAC protocols make transmission decisions based on just the carrier sense signal, which indicates that some transmission is in progress at a node. The carrier sense signal does not capture information such as who the current sender is, and how much interference this sender can cause at a particular receiver, leading to sub-optimal performance of MAC protocols that rely on carrier sense alone. Note that while information about ongoing transmissions (embedded in link-layer headers sent at the start of a transmission) is available at the PHY while a packet is being decoded, the interface does not allow this information to be passed up to the link layer until after the packet reception completes.

While using additional information from the PHY can better inform link-layer bit rate and channel access decisions, arbitrarily sharing information between the PHY and the link layer can violate the modularity that comes with layering and can break the design when the link layer or PHY evolves. Therefore, care must be taken to expose useful information between the PHY and the link layer only via well-defined interfaces. This dissertation describes how one can design architecturally clean yet better-performing link-layer protocols that are based on a new link layer-PHY interface that is better suited to the challenges of wireless communication.

1.1 PHY-Aware Link Layer Design

In this dissertation, we propose widening the link layer-PHY interface to export more information from the PHY to the link layer, enabling the design of better bit rate adaptation and MAC protocols. We propose two enhancements to the existing PHY interface:

1. **SoftPHY Hints.** The PHY passes up with each decoded bit an estimate of how confident it is about the bit, i.e., the PHY's estimate of the probability that a decoded bit is correct. We will refer to these per-bit confidences as SoftPHY hints. The SoftPHY hints of the bits in a received frame accurately reflect wireless channel quality during the frame reception, and can be used to directly compute the channel BER at the bit rate of the received frame. Our method of calculating channel BER works accurately across operating environments, independent of fading characteristics[1]. The channel BER computed from SoftPHY hints is versatile enough to be used by link-layer protocols for a variety of purposes like bit rate adaptation and channel access.

2. **Streaming PHY.** The PHY passes up parts of the decoded frame to the link layer at the granularity at which they are decoded (e.g., bits, PHY symbols), without waiting for the entire frame reception to complete. Such an interface allows the link layer to take decisions based on the current transmission on the air, and enables the design of better MAC protocols. For example, the link layer can decode the headers of ongoing receptions, identify the sender and receiver of those transmissions, and decide whether to transmit or not based on the identity of the ongoing transmissions (instead of using only the "busy" signal that indicates some transmission is in progress).

Figure 1-3 compares the new PHY interface with that in today's wireless network stacks. In addition to the design and implementation of the new interface, this dissertation presents the design, implementation, and evaluation of a rate adaptation protocol called SoftRate and a MAC protocol called SoftCMAP that are both built atop this interface. These *cross-layer* protocols achieve significant gains over existing link-layer protocols by exploiting information that is available via the new interface.

Note that we define channel BER as the probability of error of bits transmitted over the channel. Therefore, SoftPHY hints can be used to compute the underlying channel BER even using a frame that was received with no bit errors. This feature is important for some applications like bit rate adaptation, e.g., channel BER estimates of 10^{-5} and 10^{-9} at some bit rate, while both low enough to cause no bit errors in a single frame, would result in different transmit bit rate choices for the next frame. Also note that the SoftPHY hints of the received frame are used to directly *estimate* the underlying channel BER at the modulation and coding of the received frame. In contrast, channel quality metrics such as SNR of the received frame can *predict* the channel BER at any bit rate.

The concept of SoftPHY hints itself is not new; some previous work [26, 27, 33, 72] computed SoftPHY hints for a specific PHY (Zigbee or 802.15) for the purpose of identifying likely errored bits. Our design of SoftPHY hints improves upon earlier work in many ways. First, the earlier design of SoftPHY hints can only identify bit errors but cannot be

[1]Our BER computation assumes that channel noise is Gaussian, a reasonable assumption in practice.

used to compute the probabilistic channel BER (particularly when the BER is low enough to not cause any bit errors). On the other hand, our method allows the receiver to estimate the probabilistic channel BER even if the BER was low enough to not cause any bit errors in that particular received frame. Second, the earlier approach to computing SoftPHY hints works only for the specific PHY it was designed for. In contrast, our approach of computing SoftPHY hints relies on extracting and exposing internal state from the decoder of the error-correcting code in the receiver PHY; as such, our interface is compatible with most physical layers that use error correction.

1.1.1 Bit Rate Adaptation

This dissertation describes a link-layer protocol called **SoftRate** that uses the channel BER computed from SoftPHY hints as the feedback to perform bit rate adaptation. A SoftRate receiver uses the per-bit SoftPHY hints delivered by the PHY via the SoftPHY interface to accurately estimate the BER of a received frame. The SoftRate sender then uses the BER conveyed by the receiver at the current bit rate to predict the channel BER at the other rates, and before each transmission picks the bit rate that minimizes the air-time required to deliver the packet to the receiver. SoftRate also uses a heuristic to identify patterns of SoftPHY hints that correspond to transient interference, and eliminates the effect of such interference in computing the BER feedback.

Most bit rate adaptation protocols in use today [31, 37, 34, 47, 71, 11] rely on the feedback of the loss rate of link-layer frames (computed by observing which frames received a link-layer ACK and which did not) to estimate the channel quality at various bit rates, and pick a suitable rate for transmission. However, it takes multiple frame transmissions to arrive at a statistically significant estimate of the frame loss rate, which is too slow for mobile channels that change every few frames. Our evaluation of SoftRate using a combination of live experiments and trace-driven simulations shows that TCP flows achieve up to $2\times$ higher throughput when running over SoftRate as compared to when using these protocols that rely on frame reception events alone. This improvement is due to SoftRate's ability to adapt transmit bit rate at the timescale of individual frames, making it highly responsive to rapid channel variations due to mobility. SoftRate is also more robust to interference losses than frame-based rate adaptation protocols.

Other rate adaptation protocols [23, 54, 13, 29, 51] measure the SNR of the received signal using various heuristics, map the measured SNR to the channel BER at different bit rates, and then compute the optimal transmit bit rate. These protocols determine the channel BER from the measured SNR for specific modulation and coding schemes, using a channel propagation model that captures how the SNR varies with time. Therefore, the mapping between the measured SNR and the best transmit bit rate may change with the channel propagation characteristics, especially in channels with high variability that occur at vehicular speeds. The mapping from measured SNR to the optimal bit rate may also depend on the hardware being used [74], due to minor differences in the PHY implementations that result in different channel error rates for the same SNR. Therefore, bit rate adaptation protocols that use SNR must be carefully calibrated for each environment or else suffer a loss in throughput. On the other hand, the channel BER computed from SoftPHY hints accurately reflects the channel quality across all operating environments, because the channel

characteristics are implicitly captured by the SoftPHY hints. Because of SoftRate's use of this BER as the feedback for rate adaptation, SoftRate achieves between 35% to 2× higher throughput than SNR-based protocols in the absence of any environment-specific calibration. SoftRate also does not incur the overhead of pilot frequencies or known symbols that are required to estimate the SNR over a packet.

1.1.2 Channel Access

This dissertation presents a MAC protocol that improves link-layer throughput by cleverly exploiting the additional information available via the streaming PHY interface. In the popular carrier sense multiple access (CSMA) protocol, a node that has a packet to transmit first checks the carrier sense signal from the PHY and transmits only if no nearby node is transmitting. In effect, CSMA tries to ensure that only one node is transmitting at any point of time in a radio neighborhood and avoids all concurrent transmissions. In the example in Figure 1-2, CSMA correctly prevents *CS* from transmitting when *S* is transmitting, avoiding a transmission that would have resulted in a loss. However, it also prevents *ES* from transmitting successfully to *ER* when *S* is transmitting. This is called the *exposed terminal problem*[2], because *ES* and *S* are "exposed" to each and cannot transmit concurrently. The exposed terminal problem arises because the busy signal from the PHY only tells the link layer that *someone* is transmitting, not *who* the transmitter is or what impact that transmitter has on a particular receiver. Now, a streaming PHY delivers bits to the link layer as soon as they are decoded at the PHY, with the result that the link layer can decode the frame header and identify the sender and receiver of a transmission before the transmission completes. To enable concurrent transmissions between exposed terminals alone, nodes maintain a "map" of conflicting transmissions (e.g., *S* to *R* and *CS* to *CR* in Figure 1-2) and use this *conflict map* for channel access decisions. By listening to ongoing transmissions on the shared medium to identify the current transmitter, and consulting the conflict map just before it intends to transmit, each node determines whether to transmit data immediately and concurrently with an ongoing transmission, or defer till the conflicting transmission completes. Therefore, using conflict maps along with a streaming PHY improves link-layer throughput by increasing the number of successful concurrent transmissions.

To construct the conflict map, nodes optimistically assume that concurrent transmissions will succeed, and carry them out in parallel. Then, in response to the observed fate of the concurrent transmissions, they discover which concurrent transmissions are likely to work, and which aren't, dynamically building up a distributed data structure of conflicting transmissions. We show that nodes can build and maintain the conflict map in a distributed fashion by overhearing ongoing transmissions and exchanging lightweight information with their one-hop neighbors. Our protocol assumes that nodes try to perform only two concurrent transmissions at a time for two reasons—it greatly simplifies the design of the protocol, and successful concurrency between more than two nodes in a single radio neighborhood is very rarely observed in practice.

[2]Note that the exposed terminal problem should not be confused with the *hidden terminal problem* that also affects CSMA—senders *S* and *HS* in Figure 1-2 simultaneously send to *R* because they are far away from each other and therefore cannot carrier sense and avoid each other.

The streaming PHY also enables the link layer to perform a jointly-optimal bit rate adaptation and channel access decision. Observe that the exposed terminal problem becomes more complicated when we consider the presence of heterogeneous bit rates at the various senders. In the example in Figure 1-2, suppose senders *S* and *ES* transmit at the 24 Mbps bit rate to their respective receivers when sending one at a time, as indicated by some rate adaptation protocol. It is possible that the throughput at this best transmit bit rate drops for each sender when the other node starts simultaneous transmission, due to the added noise from the concurrent transmission. However, it is also possible that when both nodes send concurrently at a rate lower than the optimal, say, 18 Mbps, the additional "slack" to tolerate noise available at the lower bit rate enables them to perform concurrent transmissions successfully. In this case, it is better for the nodes to transmit concurrently at 18 Mbps, rather than share the channel at 24 Mbps and achieve an effective throughput of only 12 Mbps. To harness the throughput gains from such "partially exposed" terminals, one must perform a joint channel access and transmit bit rate decision. That is, for best system performance in this example, *S* should transmit to *R* at 18 Mbps when *ES* is transmitting, and at 24 Mbps otherwise. The streaming PHY lets a sender perform joint MAC and bit rate decisions, possibly picking different bit rates based on who else is transmitting concurrently.

We present two variations of a MAC protocol based on the idea of conflict maps. The protocols differ in how the map is computed and in the feasibility of deployment on today's commodity hardware. The first protocol, **SoftCMAP**, uses SoftPHY hints from the PHY to estimate the BER at the receiver during a concurrent transmission. Based on this BER feedback, the sender decides whether to transmit concurrently again with the ongoing transmission. Nodes also run a bit rate selection protocol similar to SoftRate, conditioned on various concurrent senders, to determine what bit rate to transmit at for concurrent transmissions. However, SoftCMAP does not lend itself to implementation on existing 802.11 commodity hardware that does not stream SoftPHY hints. The second protocol, **CMAP**, is an approximation to SoftCMAP that uses frame loss rates when transmitting concurrently to infer conflicting transmissions. The two protocols complement each other—while SoftCMAP demonstrates the full potential of the ideas in this dissertation, CMAP enables us to obtain a realistic evaluation on today's commodity hardware. We evaluate SoftCMAP using a combination of live experiments and trace-driven simulations on software radios, and CMAP using a real implementation and deployment on commercial 802.11 hardware. Our evaluation shows that identifying and exploiting exposed terminals leads to a near doubling of throughput for the exposed senders, and improvements of up to 50% in aggregate network throughput in some common topologies.

1.2 Contributions and Roadmap

This dissertation makes the following contributions.

1. **New link layer-PHY interface.** We observe that many problems with today's link-layer protocols can be attributed to insufficient information about the wireless channel at the link layer. With the unique challenges of wireless communication in mind,

we augment the link layer-PHY interface in today's wireless network architecture with two new mechanisms: the PHY exports SoftPHY hints or per-bit confidences with each decoded bit, and the PHY streams received bits to the link layer as soon as they are decoded and before the entire frame reception completes. The SoftPHY hints of a received frame can be used to estimate the channel BER during the frame reception without making any assumptions on the channel propagation characteristics. The channel BER computed from SoftPHY hints is accurate and can be used by link-layer protocols to make better transmit bit rate and channel access decisions. The streaming interface between the link layer and the PHY enables the link layer to learn about the current transmission on the air and make smarter channel access and transmit bit rate decisions based on the knowledge of ongoing transmissions. Our design of these interfaces is general enough to be applicable to a wide class of physical layers.

2. **PHY-aware protocols for bit rate adaptation and channel access.** We show how one can design better link-layer protocols by using the additional information available via the streaming SoftPHY interface. The SoftRate protocol performs fast and accurate transmit bit rate adaptation using the channel BER estimated from SoftPHY hints as the feedback from the receiver to the sender. It achieves between 35% to $2\times$ higher throughput than existing rate adaptation protocols. The SoftCMAP and CMAP protocols are MAC protocols built atop the streaming PHY interface. These protocols use a map of conflicting transmissions and make transmit decisions based on which transmissions are ongoing, instead of using the carrier sense signal from the PHY. Our evaluation of these protocols show that they improve aggregate network throughout by up to 50% in common network topologies by increasing the number of successful concurrent transmissions.

The rest of the dissertation is organized as follows. Chapter 2 explains the basics of wireless communication and provides an overview of the PHY and link layers in today's network architecture. Chapter 3 describes our design and implementation of the SoftPHY interface. Chapter 4 describes the design, implementation, and evaluation of the SoftRate protocol. Chapter 5 describes the SoftCMAP and CMAP protocols. Chapter 6 concludes the dissertation.

Chapter 2

Background

In this chapter, we describe some basic concepts of wireless communication systems that are useful in understanding the rest of the dissertation. We first describe the characteristics of the wireless channel that make link-layer protocol design challenging. We then describe the PHY and link layers in current communication systems in order to understand the context for the new cross-layer protocols proposed in this dissertation.

2.1 The Wireless Channel

The wireless channel propagates electromagnetic waves at the carrier frequency of the communication system from the transmitter to the receiver. Along the path, the wave experiences channel-induced effects that distort the signal in many ways. Understanding these changes is important for protocol designers, because higher-layer protocols (bit rate adaptation protocols in particular) often need to account for them in the design decisions. The wireless signal suffers four types of changes during propagation: *attenuation*, *shadowing*, *fading*, and *interference*. Suppose $s(t)$ denotes the transmitted signal. Then the received signal $r(t)$ can be expressed as:

$$r(t) = h(t)s(t) + n(t) \tag{2.1}$$

where $h(t)$ is the channel gain at time t (determined by attenuation, shadowing, fading, and interference) and $n(t)$ is the random noise added by the channel. The channel noise is often approximated as a zero-mean Gaussian random variable. The noise is also considered to be "white", i.e., it has a flat spectrum and is similar across all frequencies. A channel with constant gain can be approximated as an Additive White Gaussian Noise (AWGN) channel.

Attenuation and shadowing. Every electromagnetic wave suffers a loss in power according to the inverse square law as it travels in free space. A similar effect applies to electromagnetic waves carrying data—signal powers decays at the rate r^{α}, where r is the distance from the source and α is the path loss exponent. Unlike propagation in free space, the path loss exponent, α, is often greater than 2 in real-life environments. Signals also suffer loss in power due to obstacles along the way, an effect called shadowing. Attenuation and shadowing are slow-changing effects, and do not cause much variability in the

signal unless the distance between the sender and the receiver or the placement of obstacles change. Therefore, the variability caused by these effects is somewhat easier to handle at the receiver. For example, bit rate adaptation protocols only need to adapt the rate at a coarse timescale (say, every few seconds) in order to handle changes due to attenuation or shadowing.

Fading. When a signal is transmitted, multiple copies of it propagate via different paths to the receiver and therefore suffer different channel-induced changes. These copies of the signal, which differ slightly from each other in amplitude, phase, and frequency, can combine constructively or destructively at the receiver and cause an apparent variation in the transmitted signal. This variation in the signal caused due to multipath propagation effects is commonly referred to as multipath fading. The fading loss of the signal is superimposed over other channel losses due to attenuation and shadowing. We will now describe the nature of fading-induced channel variations and its implications for bit rate adaptation.

The multiple copies of the signal that interact at the receiver can differ from each other in two ways:

- In a rich scattering environment, the copies of the signal can be spread out in time due to different copies of the signal traveling along paths of slightly different lengths and arriving at different times at the receiver. The maximum duration between the time of arrival of the first and last copies of the signal is called the *delay spread* T_D of the signal.

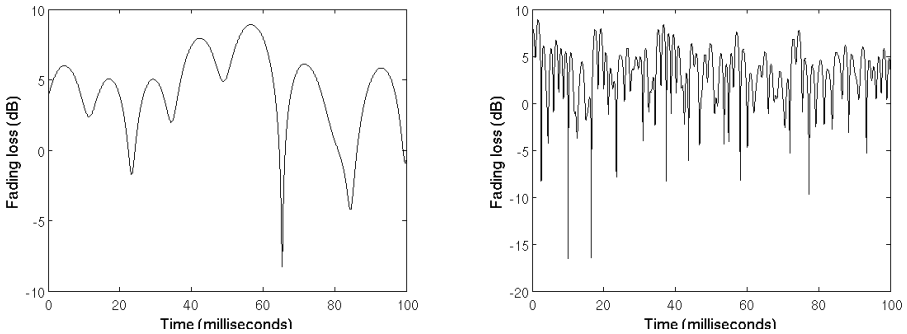
- When the sender, receiver, or any of the scattering objects in the environment are moving, the copies of the signal suffer varying amounts of Doppler shift in their frequency due to the Doppler effect. This spread in the frequency among the copies of the signal is called the *Doppler spread*, f_D, of the signal. The Doppler spread depends on the speed of movement, v, the carrier frequency, f, and the speed of light, c, as follows:

$$f_D = \frac{vf}{c} \tag{2.2}$$

When multiple copies of the signal that are spread out in time interact, the effect is an apparent variation of the channel gain across the frequency band of the signal. The theory behind this phenomenon is beyond the scope of this dissertation, and we refer the interested reader to [50]. The larger the delay spread of the signal, the smaller the frequency over which the channel stays the same. The *coherence bandwidth*, B_c, is defined as the width of the frequency band across which the channel fading effects remain the same, and is inversely related to the delay spread:

$$B_c \propto \frac{1}{T_D} \tag{2.3}$$

If the coherence bandwidth of a system is smaller than the width of the transmission band, then the different frequencies in the transmitted signal suffer different amounts of channel losses. Such a system is said to have *frequency-selective fading*, which is a hard problem to tackle in communication systems—the receiver must separately estimate the



(a) Coherence time = 10 ms. (b) Coherence time = 1 ms.

Figure 2-1: Channel losses during slow fading: a 100 ms snapshot of the fading losses in simulated fading channels with coherence time (a) 10 ms, and (b) 1 ms.

channel gains at the various frequencies and compensate for it. Frequency-selective fading can be tackled by modulating different frequencies at different bit rates [51], i.e., by dividing a channel into several subcarriers and sending data at different bit rates on each subcarrier. While the bit rate adaptation protocol in this dissertation adapts bit rate with time, our ideas are applicable to the problem of rate adaptation across frequency as well.

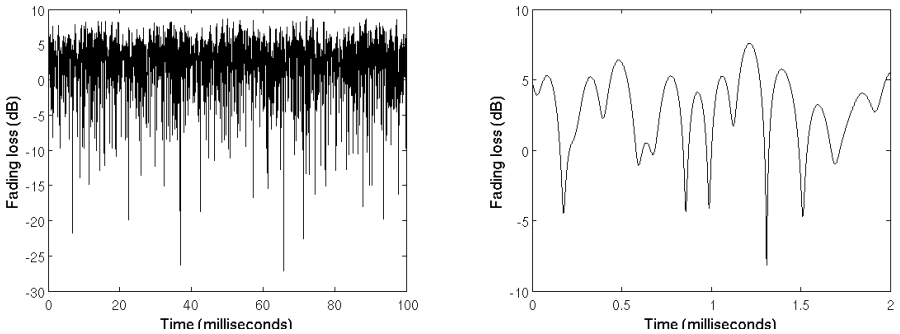
On the other hand, when multiple copies of the signal that are spread out in frequency interact, the effect is of apparent variation of the channel gain across time. The *coherence time* of the channel, T_c, is defined as the period of time over which the channel effects can be assumed to be correlated. That is, two signals separated in time by more than T_c will suffer uncorrelated fading losses. The coherence time of the channel is related to the Doppler spread in frequency as follows [50]:

$$T_c \approx \frac{0.4}{f_D} \tag{2.4}$$

That is, the faster the speed of movement, the greater the temporal variation of the signal, and the lower the coherence time. Note that Equation 2.4 is only a theoretical approximation, and the actual coherence time of any given channel can only be obtained by real-world measurements.

For pedestrian mobility, the Doppler spread in frequency is typically tens of Hz, and the coherence time is roughly hundreds of milliseconds. For example, a walking speed of 0.5 m/s (1.8 mph) on a 2.4 GHz carrier (the ISM band used by 802.11g/n) results in a Doppler spread of 4 Hz and a coherence time of 100 ms. Note that typical packet transmission times in today's wireless data networks are up to a few milliseconds. As a result, the channel stays the same for many packet durations at walking speeds. This scenario is often referred to as *slow fading*. In slow fading channels, the receiver can estimate the channel quality at the beginning of a packet and assume it stays the same for the next few packets.

For vehicular mobility, the Doppler spread is tens to hundreds of Hz, and the channel coherence time is few tens of milliseconds, i.e., a few packet durations. The fading losses generated by a simulated fading channel model over a 100 ms snapshot for channel coher-



(a) Snapshot over 100 ms. (b) Snapshot over 2 ms.

Figure 2-2: Channel losses during fast fading: fading losses in a simulated fading channel with coherence time of 100 μs at timescales of (a) multiple packet durations, and (b) single packet duration.

ence times of 10 ms and 1 ms are shown in Figure 2-1[1]. One can see that a fading channel goes into a *deep fade*, or a period of heavy fading loss and high error rate, every once in a while. The frequency of these fades depends on the coherence time of the channel. One can also see from the figure that multipath fading requires bit rate adaptation protocols to react very quickly to channel changes. For example, when the sender or the receiver is in or near a moving vehicle, the rate adaptation protocols may have to pick a new transmit bit rate once every ten or twenty milliseconds, i.e., once every few packets.

For very fast mobility like train speeds, the Doppler spread is a few tens of KHz, resulting in coherence times smaller than a packet duration, usually few hundreds of microseconds. As a result, the channel changes widely even within a single packet duration. Such channels are called *fast fading* channels. Fast fading channels with coherence time as small as 300 μs have also been observed in urban environments with fast moving vehicles [13]. Figure 2-2 shows the channel losses for a simulated fading channel at train speed mobility at two different timescales corresponding to multiple packet durations and a single packet duration. Tracking the channel quality of a fast fading channel for rate adaptation is a very hard problem, because one must sample the channel quality very frequently and many times within a single packet.

Interference. Because the wireless channel is a shared medium, multiple sources transmitting signals at the same time can result in the signals being corrupted and undecodable at the corresponding receivers. This phenomenon is called interference. Interference is generally handled by channel access protocols at the link layer. A good channel access protocol will try to ensure that interference does not occur by enabling concurrent transmissions only when such transmissions are likely to succeed. When packets are lost due

[1]Because fading occurs due to the interaction of a large number of copies of a signal that cannot each be precisely measured, the fading effects are not deterministic (unlike attenuation) and are randomly distributed with some statistical properties. For scattering environments that do not have a line-of-sight propagation, the Rayleigh distribution approximately models the fading loss [50]. We use a simulated Rayleigh fading channel model to generate Figures 2-1 and 2-2.

to a collision, channel access protocols take appropriate measures (like sender backoff) to ensure the interference does not persist.

Wireless channel capacity. The wireless signal from a source to the destination suffers from the channel induced losses of attenuation, shadowing, fading, and interference as described above, leading to the receiver incorrectly decoding parts of the signal. As a result, the message recovered at the receiver has bit errors and is different from the message transmitted. However, one can use a variety of techniques like error correction to make the signal robust to distortion, thereby making the communication more reliable, though less efficient in terms of the rate of transmission of information.

The signal to noise ratio (SNR) is a measure of how strong the sender's signal is at the receiver compared to the ambient noise on the channel, i.e., from Equation 2.1, the SNR at time t is the ratio of $h(t)s(t)$ to $n(t)$. The higher the SNR of the received signal, the easier it is to separate the signal from the background noise, and the lower is the bit error rate (BER) of the decoded message at the receiver. Losses due to channel effects such as attenuation and fading decrease the channel gain $h(t)$ and hence the SNR of the transmitted signal, causing bit errors. Shannon's work on information theory provides a way to bound the rate of communication of a wireless channel given two parameters: the SNR of the source at the destination, and the bandwidth of the channel B available for the communication. The capacity of the channel, defined as the maximum achievable rate of reliable communication, is given by:

$$C = B \log_2(1 + \text{SNR}) \tag{2.5}$$

Note that Shannon's equation only specifies what the maximum achievable rate of transmission of information is, but does not specify how to achieve that rate. Communication systems try to achieve communication at Shannon's capacity using various techniques that improve the reliability of the transmission.

2.2 The Physical Layer

The physical layer is concerned with the transmission and reception of digital information across the wireless channel via an RF front-end. Figure 2-3 shows a simplified transmitter and receiver pipeline in the 802.11a/g PHY, the PHY used in the various prototype implementations in this dissertation.

The transmitter first scrambles a message, by multiplying it with a pseudo-random bit sequence, in order to eliminate unwanted regularity in the message that may confuse some PHY algorithms. Next, the transmitter adds some form of forward error correction (FEC) to the message bits in order to protect against channel-induced distortions. To make FEC more effective, a transmitter optionally interleaves the coded bits, so that adjacent coded bits during decoding would have been temporally non-adjacent and therefore unlikely to be in error all at once due to the burst errors causes by a deep channel fade. The transmitter then modulates the bits. The modulator or mapper takes groups of bits (coded bits when FEC is used) and maps them into complex baseband *symbols* for transmission by the RF

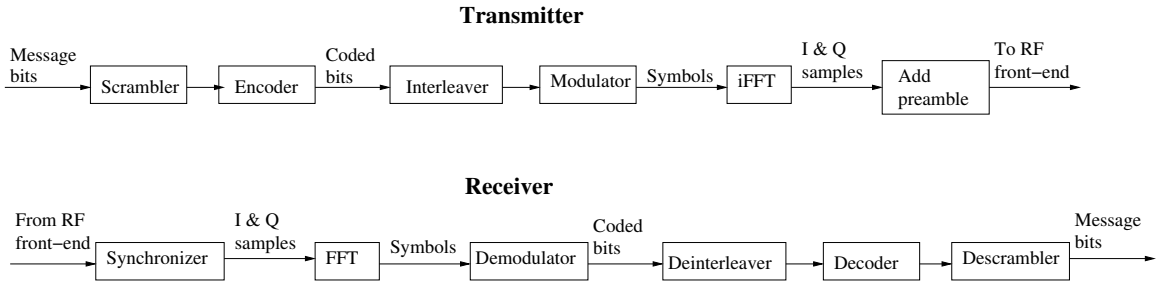


Figure 2-3: Simple transmitter and receiver pipelines for 802.11a/g-like physical layers.

front-end. The complex symbols produced by the modulator correspond to the amplitude and phase of the sinusoid of a particular frequency. However, transmission hardware operates only on time samples. The transmitter therefore converts the symbols produced by the modulator into time-domain samples using the inverse Fast Fourier Transform (iFFT) operation. The output of the iFFT is complex time-domain samples with real (I) and imaginary (Q) components. The I and Q components represent the two orthogonal dimensions along which information can be transmitted using electromagnetic waves. The transmitter then appends a known symbol pattern to the beginning of the frame, called the preamble, to enable detection of the start of the transmission at the receiver. The preamble is followed by a PHY header, often sent at the lowest base bit rate, that tells the receiver what coding rate and modulation to use for the rest of the frame. These *baseband* I and Q samples are then *upconverted* into the desired transmission frequency and transmitted as analog waves by the RF front-end.

The receiver PHY undoes the various operations of the transmitter. The receiver constantly looks for the known preamble in the RF samples streaming in. When it synchronizes with a preamble, it estimates the effects of channel-induced changes (like the channel gain) and other RF-induced changes (like the frequency offset between the two RF front-ends) on the signal using the known preamble, and corrects for them in the rest of the signal. The receiver then reverses the transmission process: it performs an FFT to convert the time-domain I and Q samples into frequency-domain symbols, demodulates them into coded bits, deinterleaves and decodes the bits, and finally descrambles the output of the decoder to recover the original message.

The modulation and coding schemes used together determine the bit rate of the PHY transmission. Different modulation schemes differ in the number of bits in one symbol, and in the set of valid symbols available for transmission. For example, in the Binary Phase Shift Keying (BPSK) scheme, the bits "0" and "1" map to $+1$ and -1 on the real line respectively and the imaginary component is not used. For the Quadrature Phase Shift Keying (QPSK), the bits "00", "01", "10", and "11" could map to the symbols $\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}j$, $-\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}j$, $\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}j$, and $-\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}j$ on the unit circle. Another example of a popular modulation scheme is the Quadrature Amplitude Modulation (QAM) with 16, 64, or 256 valid symbols in a unit square. A diagram showing all the valid symbols in a modulation scheme is called the constellation diagram. Figure 2-4 shows the constellation diagrams of various modulation schemes used in 802.11a/g/n. Note that we show the constellations as lying within a unit circle or square only as a way of normalization; the signal is scaled by

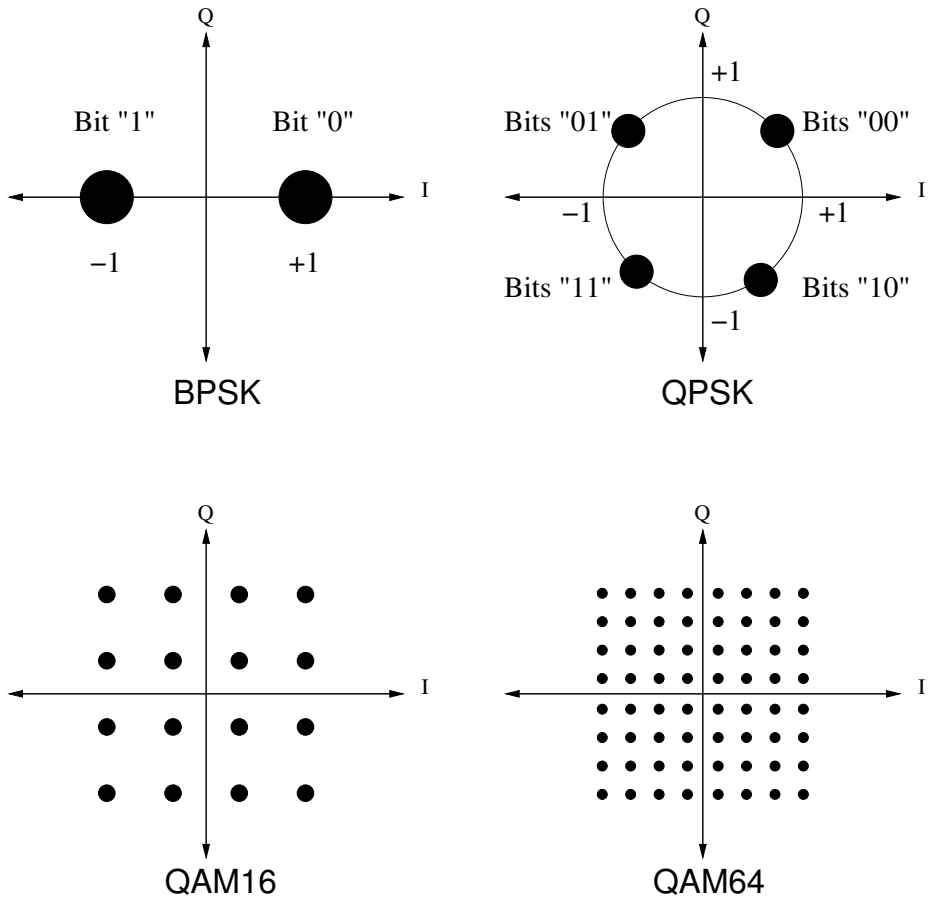


Figure 2-4: Constellation diagrams for the BPSK, QPSK, QAM16, and QAM64 modulation schemes that are used in 802.11a/g/n.

the square root of the transmit power before transmission.

The bit rate also depends on the FEC used. The *rate* of an FEC code is the rate at which original information is transmitted via the coded bits. If a code takes m bits of the original message and transmits them as $n \geq m$ coded bits, then the rate of the code is m/n. There are two main types of FEC codes: block codes and convolutional codes. Block codes operate on fixed-size chunks of the original message and convert them into coded chunks with more bits. A rate m/n block code takes a chunk of m bits in the original message and maps them into an n-bit codeword for transmission. The decoder maps each received codeword to the "closest" (as measured by Hamming distance) valid codeword and recovers the original transmitted bits. Convolutional codes, on the other hand, operate on streams of bits of arbitrary length. A convolutional code has an additional parameter, its *constraint length*, L, that denotes how many past information bits are used to produce the current coded bit. A rate m/n convolutional encoder takes in m input bits and produces n coded bits which depend on the input bits of the last L stages. The most common decoder used to decode convolutional codes is the Viterbi decoder [66, 28]. The Viterbi decoder examines the entire stream of received coded bits, and recovers the *most likely* sequence of inputs that could have generated the coded output bits.

Our PHY prototypes in this dissertation implement the Orthogonal Frequency Division Multiplexing (OFDM) PHY, commonly used in high throughput physical layers like 802.11a/g/n, WiMAX, and LTE. In OFDM, the available frequency is divided into multiple sub-channels or subcarriers, and each is used to independently transmit a stream of modulated I and Q samples. There are many advantages to using OFDM over transmitting data over a single frequency band, including lesser susceptibility to frequency-selective fading and simpler algorithms for equalization.

Link layer interface. The interface between the PHY and the link layer in today's network stack is at the frame granularity, mostly because it was adapted from its wired counterpart. Even though the PHY operation is pipelined in hardware implementations and the bits are decoded as they are received, the PHY waits for the frame reception to complete before passing them up to the link layer. In addition to received frames, the PHY exports to the link layer an estimate of SNR of the reception. The SNR is typically estimated from measuring the distortion to known symbols in a packet. For example, one can use the distortion to the preamble at the start of the packet, or to the pilot symbols sent in a separate pilot channel throughout the packet. The PHY also performs carrier sense or clear channel assessment (CCA): the carrier is considered busy if the energy on the channel exceeds a threshold or if a preamble has been detected recently.

This dissertation proposes widening the interface to the link layer to export additional information that can improve the performance of link-layer protocols. With the new interface, PHY streams bits to the link layer as soon as they are produced by the pipeline. As we describe in Section 3.2, the decoders contain information about how close the received signal was to the transmitted signal. This information can be used to compute the probabilities that the decoded bits are correct, which are then exported to the link layer as SoftPHY hints. These SoftPHY hints are subsequently used by link-layer protocols to make transmit bit rate or channel access decisions.

2.3 The Link Layer

The link layer at the transmitter receives frames from the network layer. It adds link layer-specific header information, decides what bit rate to send the frame at (bit rate adaptation), decides what the right time to make the transmission is (channel access), and passes the frame to the PHY for transmission. The receiver link layer receives the frame from the PHY and checks if the frame has an any errors, and arranges for retransmission from the transmitter in case of errors (error recovery). While link layers in different communication systems differ on the exact details, the broad functions of the wireless link layer remain the same: error recovery, channel access, and bit rate adaptation.

Error recovery. The PHY uses FEC to recover from bit errors, but may not succeed in doing so always. As a result, the frame delivered to the link layer may still contain errors. In order to improve the reliability of communication, the link layer also attempts to recover errors before passing the frame to the higher layers. The most common type of error recovery is frame retransmissions, as used in the 802.11 protocol. The link layer

Modulation	Code Rate	802.11 Throughput
BPSK	1/2	6 Mbps
BPSK	3/4	9 Mbps
QPSK	1/2	12 Mbps
QPSK	3/4	18 Mbps
QAM16	1/2	24 Mbps
QAM16	3/4	36 Mbps
QAM64	1/2	48 Mbps
QAM64	2/3	54 Mbps

Table 2.1: Combinations of modulations and coding rates used in 802.11a/g OFDM, and the raw throughput achieved over a 20 MHz channel.

embeds a CRC over the contents of the frame, and checks the CRC on receiving a frame from the PHY. If the CRC matches, the link layer sends an ACK to the transmitter. In the absence of an ACK, the transmitter retransmits the frame till it receives an ACK or for a fixed maximum number of times before giving up.

However, note that most frames that are in error typically have a small number of bits in error, and retransmitting the entire frame is a waste of channel capacity. As a result, smarter error recovery schemes, called "Hybrid ARQ", have been proposed in the research literature. In general, the term Hybrid ARQ refers to any scheme that combines forward error correction (FEC) and automatic repeat request (ARQ). Systems such as WiMax [25], certain satellite channels [39], cellular high-speed packet access (HSPA), cellular LTE, and more recent proposals such as ZipTx [38] use a form of hybrid ARQ called *incremental redundancy* [41, 43]. Incremental redundancy forgoes aggressive FEC on the first transmission of a packet, requesting subsequent transmissions of parity bits with ARQ only if needed.

All the above error recovery protocols rely on the PHY passing up received frames alone. Recent research has explored cross-layer approaches to error recovery, where the PHY passes up additional information about received frames to the link layer. Partial packet recovery (PPR) [27] is an error recovery scheme that uses SoftPHY hints exported from the PHY to retransmit (mostly) only those bits believed to be in error. While we share the notion of per-bit confidences with this earlier work, the SoftPHY hints proposed by the authors are different from and less general than the SoftPHY hints in this dissertation.

Bit rate adaptation. Communication systems often have the capability to transmit data at multiple bit rates, where each bit rate achieves a certain rate of transmission of information by using a particular choice of modulation and coding rate. For example, the 802.11a/g OFDM system can transmit data at the rates of 6, 9, 12, 18, 24, 36, 48, and 54 Mbps using different combinations of modulation and coding rates, as shown in Table 2.1. Higher bit rates can be used on links with higher SNR and vice versa. However, the maximum rate that can be used is limited by Shannon's capacity of the channel at that SNR. The SNR of a link and hence the suitable bit rate change with time in a time-varying channel. As a result, the link layer must make the decision of what bit rate to use dynamically based on

its estimate of the current channel quality, in order to pick the bit rate that is closest to what the channel can sustain at that quality.

Note that the problems of error recovery and bit rate adaptation are orthogonal to each other. While error recovery schemes improve capacity in a time-varying wireless channel, their performance is still contingent on choosing appropriate bit rates for individual transmissions. In other words, while error recovery chooses *which data* to transmit, rate adaptation chooses *which bit rate* to transmit the data at.

The link layer requires information about current channel quality to perform bit rate adaptation. There are two ways of estimating channel quality using the information delivered via the current PHY interface: estimating the frame loss rate at each bit rate from the fraction of frames that elicit a link-layer ACK, or estimating channel BER at various bit rates from the SNR estimate provided by the PHY. Section 4.1 surveys the existing rate adaptation protocols that use one of the above two types of channel quality feedback, and identifies their shortcomings. This dissertation uses a novel feedback for rate adaptation, the BER computed from SoftPHY hints, to perform better rate adaptation.

Channel access. A good MAC protocol fairly arbitrates access to the shared medium among the competing transmitters without causing too much interference to any transmission, while utilizing the capacity of the channel as efficiently as possible. Channel access protocols arbitrate access in the time as well as frequency domains, i.e., they specify what time transmissions must start at and what frequency sub-channels should be used for the transmission.

Arbitrating access to the medium does not necessarily mean allowing only one transmission to proceed at a time, because the channel conditions may be such that more than one transmission can be sustained without causing interference to any of the transmissions. For example, the set of available bit rates in a system may be such that the most suitable bit rate is actually lower than the channel capacity, while the immediately higher rate is higher than the channel capacity and hence unsustainable. That is, the best transmit bit rate has enough "slack" in its signal to tolerate additional noise or interference without suffering bit errors. In such cases, the channel access scheme may decide to have an additional transmission that adds a small amount of interference but does not impact the quality of the original link in a significant way. A good channel protocol schedules one or more transmissions along the time and frequency dimensions, with the goal of maximizing aggregate network throughput.

MAC protocols are of two types: contention-based and non contention-based. In non contention-based schemes, a centralized scheduler decides which transmissions must proceed when. Two popular examples are the Time Division Multiple Access (TDMA) and Frequency Division Multiple Access (FDMA) schemes, where the scheduler allocates timeslots or frequency subcarriers respectively to various competing transmitters with the goal of maximizing system throughput. Non contention-based protocols are often used in cellular systems, where the flows are usually constant bit-rate with stringent Quality of Service (QoS) requirements. However, such schemes are sub-optimal for data-based networks where the service is best effort, and the traffic pattern is unknown apriori and bursty by nature. Data networks mostly use contention-based channel access schemes, where transmit-

ters compete and figure out who should transmit in a distributed fashion. The contention-based schemes have the advantage that only a subset of all transmitters that have data to send can compete at any time and utilize the channel efficiently. However, it is also harder to coordinate between senders and schedule concurrent transmissions due to the absence of a centralized scheduler.

The most popular contention-based channel access protocol used in data networks today is Carrier Sense Multiple Access (CSMA). With CSMA, transmitters use the carrier sense information available via the PHY interface to transmit only when the carrier is idle. CSMA suffers from the exposed and hidden terminal problems as illustrated in Figure 1-2. Most modifications to improve CSMA add extra signaling frames that reserve the medium before a transmission (e.g., RTS and CTS frames [32]), allowing nodes to virtually carrier sense the medium and coordinate transmissions using information from the reservations. Section 5.6 surveys such mechanisms in more detail. However, such approaches increase the control overhead and lower link-layer throughput. Other solutions to the hidden terminal problem involve using PHY techniques to simultaneously decode both the frames in collisions resulting from imperfect carrier sense [19]. This dissertation proposes new channel access protocols that address the exposed terminal problem of CSMA and increase the number of successful concurrent transmissions in the network. Our protocols coordinate concurrent transmissions between senders without adding additional control overhead or introducing new signaling frames. Instead, the streaming PHY interface makes information about ongoing transmissions available at the link layer faster, allowing nodes to coordinate and transmit concurrently with exposed senders.

Chapter 3

Estimating Wireless Channel BER Using SoftPHY Hints

Estimating the quality of the wireless channel quickly and accurately is useful for a number of link and higher layer functions that depend on the instantaneous channel quality. For example, bit rate adaptation [31, 37, 34, 47, 71, 11, 23, 54, 13, 29] or transmit power adaptation protocols [59, 69, 6, 62] pick the transmit bit rate or power level based on how good the channel conditions to a destination are. Some link-layer protocols [51, 15, 20] also allocate transmit frequencies based on the channel quality of various users across different frequencies. Researchers have also proposed scheduling packets for transmission from the link-layer queue based on the channel quality to various destinations, picking the transmission that is most likely to succeed [9]. Some protocols [68, 33] also perform channel access decisions based on the quality of various links in the presence of concurrent transmissions.

The most fundamental measure of wireless channel quality is the channel bit error rate (BER), because the BER determines the frame loss rate or throughput or any of the other higher-layer metrics of interest. The channel BER is the probability of a bit error in transmissions over the channel. For a long stream of bits, assuming unchanged statistics, the channel BER is also the fraction of bits that are in error. The information delivered via the PHY interface today cannot be used to measure BER directly. Instead, protocols at the link and higher layers use a variety of other metrics to estimate channel quality.

Some link-layer protocols measure channel quality using purely link-layer information. For example, most bit rate adaptation protocols measure the loss rate of link-layer frames (computed as the fraction of frames that fail to elicit an ACK frame from the receiver), and use this as a proxy for channel quality. Some link-layer error recovery schemes [38] stuff pilots bits into the link-layer frame, and count how many of them were received incorrectly to estimate channel BER. We note that all these methods are somewhat cumbersome and inefficient in estimating channel quality. Frame loss rate is a very coarse grained metric because one requires many frame transmissions to converge to a meaningful estimate. The technique of stuffing pilot bits does not calculate the *true* channel BER on every frame; for example, if the channel BER is low and and none of the 100 pilot bits in a frame is in error, then one does not know if the channel BER is 10^{-3} (i.e., the pilot bits barely made it through) or 10^{-9} (i.e., the channel quality is really good). For these reasons, using just link-layer information is not very effective in estimating the BER of wireless channels that

vary quickly with time.

Most protocols today use the signal-to-noise ratio (SNR) of the received signal as a measure of channel quality. The average SNR of the received signal is approximately estimated using the distortion to known symbols in the packet (Section 3.1.1), and then mapped to the expected channel BER using analytical expressions or empirical measurements. The SNR of a received packet can be used not only to estimate the channel BER at the bit rate of the received packet, but also to predict the channel BER at other bit rates. However, in time-varying channels where the SNR is not constant during a packet, the average SNR alone is not sufficient to characterize how the SNR varies over the entire packet. In other words, one needs a model of the propagation channel in order to determine the BER at various bit rates from the measured average SNR (Section 3.1.2). As such, link-layer protocols using SNR information must be trained for a particular operating environment.

In contrast to existing approaches, this dissertation proposes a method to estimate the channel BER *directly* from SoftPHY hints, without requiring a knowledge of the channel fading characteristics. Our method uses just one frame reception to estimate the channel BER at the bit rate of the received frame, making it suitable for link-layer protocols that need accurate and responsive channel estimation. SoftPHY hints can estimate the underlying probabilistic channel BER even using a frame that has no bit errors. Our method relies on using information that is easily available at the physical layer: *soft output* decoders for error correcting codes also compute the *log-likelihood ratios* for each decoded bit. These ratios, which the PHY exports as SoftPHY hints, indicate how likely it is that an output bit is correct. Therefore, they implicitly capture the quality of the time-varying channel and can be used to compute the expected probability of bit error on the channel. We describe our design of SoftPHY hints and how they are used to compute channel BER in Section 3.2. Our approach works for any PHY that uses a linear block or convolutional code, and hence is general enough to be applicable to a wide range of physical layers (Section 3.2.2).

We observe that the channel BER computed from SoftPHY hints is a less general metric of channel quality than SNR, because it characterizes the channel only at the particular modulation and coding of the received frame. However, it is a more useful feedback metric for link-layer protocols, because one can easily determine the channel BER and link-layer throughput without any additional knowledge of the environment (Section 3.2.1).

Note that the Hamming distance between the received codewords and the closest valid codewords has also been suggested as a candidate for SoftPHY hints for PHYs using block codes in previous work in the context of error recovery [26, 27, 33, 72]. While these SoftPHY hints based on Hamming distance are quite useful to identify bit errors without knowing the actual payload, they do not correspond to the probabilistic channel BER in any meaningful way and are orthogonal to the BER estimation problem being considered in this chapter (Section 3.2.3).

We implement a PHY capable of exporting SoftPHY hints both using software radios and an FPGA-based hardware platform (Section 3.3). We show that computing SoftPHY hints adds only a small overhead in terms of implementation complexity and processing latency. We collect packet traces with SoftPHY hints using our prototypes and show that the channel BER estimated from SoftPHY hints is accurate across a wide range of wireless propagation environments (Section 3.4). The subsequent chapters in this dissertation present two link-layer protocols, SoftRate and SoftCMAP, that use this BER estimate from

SoftPHY hints to perform rate adaptation and channel access.

3.1 Related Work

SNR is the most common metric of channel quality in a number of wireless protocols. The SNR of a received signal measures the strength of the sender's signal at the receiver relative to the background noise on the channel. This section describes some common methods of estimating the SNR of the received signal, and ways to map this SNR to the channel BER.

3.1.1 Measuring SNR

Measuring SNR once per frame. In slow fading channels that do not vary much over the duration of a single frame, the SNR can be measured once per frame using a variety of methods. A common approach is to estimate the SNR from the distortion of the received preamble as compared to the expected preamble at the start of the packet [56]. Clearly, the lower the SNR, the greater the distortion in the received preamble. Another approach is to estimate SNR using the change in the AGC (Automatic Gain Control) when a packet reception begins, because the AGC adjustment depends on the signal strength of the received signal. Such methods of estimating SNR incur very little overhead, and are commonly used in existing commodity WLAN hardware.

Computing average SNR from pilot subcarriers. Measuring SNR once per frame will be inaccurate in fast fading channels, because the SNR is not constant for the duration of the frame (see Figure 2-2). In such channels, one must measure the SNR continuously to compute an *average SNR* over the entire frame. One way to track the SNR continuously over a frame is to send known symbols or *pilots* simultaneously over a separate channel, and use the distortions of the pilots to estimate SNR. SNR estimation using pilots is commonly used in cellular PHYs. We now describe a specific example of such an SNR computation in the context of an OFDM PHY [51]. This technique places known pilot symbols over a subset of OFDM subcarriers (called pilot subcarriers), and uses the reception of these pilots to track the SNR throughout the packet.

The SNR γ can be written in terms of the received power R and noise power N_0 as follows.

$$\gamma = \frac{R - N_0}{N_0} = \frac{R}{N_0} - 1 \tag{3.1}$$

Now, the received signal r, channel gain h, transmitted signal s, and the channel noise n on a particular symbol are related by the following equation:

$$r = hs + n \tag{3.2}$$

For each received pilot symbol, one can obtain an estimate for the noise as follows:

$$n = r - hs \tag{3.3}$$

Note that r, h, and s are known at the receiver's demodulator. Now, we can compute the value of n for each of the pilot tones over the entire packet, and use these values to compute the average noise power.

$$N_0 = E(n^2) \tag{3.4}$$

We can also compute the average received power from the received symbols.

$$R = E(r^2) \tag{3.5}$$

Now from equations 3.1, 3.4, and 3.5, we can compute the average SNR during the packet reception. If the number of OFDM symbols in a packet is N, the number of pilot subcarriers per OFDM symbol is T, and the total number of subcarriers in an OFDM symbol is C, then we average the noise power over $N \cdot T$ samples to compute N_0, and the received power over $N \cdot C$ samples to compute R. One can also compute the SNR over chunks of the packet by averaging the signal and noise from pilots in that chunk alone.

Estimating average SNR in this manner incurs the additional overhead of transmitting and receiving pilots when compared to the method of estimating SNR using the preamble. The estimation error may also be high if the number of pilots is small. However, this method is more accurate in capturing channel quality when the channel changes significantly during a frame transmission duration.

3.1.2 Mapping SNR to BER

We describe how one can use an SNR measurement to estimate the channel BER in different channels. We will first consider simple uncoded AWGN channels, and then extend the approach to cover fading and channel coding. Our discussion below explains why BER estimation from measured SNR depends on the channel propagation characteristics in realistic coded fading channels.

Uncoded AWGN channels. Let E_b denote the transmission energy per data bit on the channel, and N_0 denote the noise power in the channel. Then the SNR per bit γ_b of the transmitted signal is given by:

$$\gamma_b = \frac{E_b}{N_0} \tag{3.6}$$

Protocols using SNR to estimate link-layer performance rely on estimating the transmission SNR, and then mapping the SNR to the expected channel BER using an *SNR-BER curve*. That is, protocols use a function for the probability of bit error P_e given the measured SNR per bit γ_b. The SNR-BER curve depends on the modulation and coding schemes being used—schemes that pack more redundancy into the signal to make it robust to noise achieve a lower BER for the same SNR. We now describe the computation of the SNR-BER curves for the BPSK and QPSK modulations in transmissions over the AWGN channel with no channel coding.

The BPSK modulation transmits one of two information symbols to represent the bits "0" and "1" (see Figure 2-4). Let us denote the possible transmission symbols as s_1 and s_2.

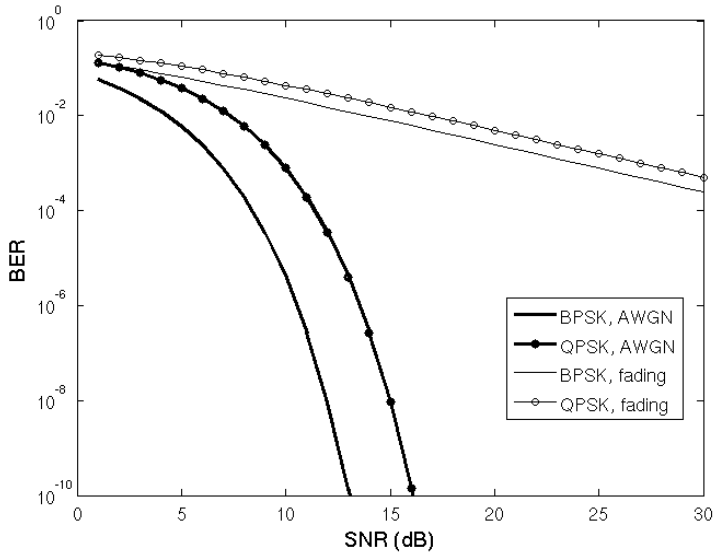


Figure 3-1: Theoretical relationships between channel BER and SNR per bit of the transmitted signal, for BPSK and QPSK modulations in AWGN and Rayleigh fading channels. The functions plotted are shown in equations 3.10, 3.11, 3.14 and 3.15.

Recall that E_b denotes the transmission energy per bit and that the amplitude of the transmitted signal is proportional to the square root of the energy. Therefore, the transmitted symbols are at $+\sqrt{E_b}$ and $-\sqrt{E_b}$. Suppose the transmitted symbol is $s \in \{s_1, s_2\}$. Then the received symbol r is given by

$$r = hs + n \tag{3.7}$$

where h is the channel gain and n is Gaussian white noise with variance $\sigma_n^2 = \frac{1}{2}N_0$. Note the $\frac{1}{2}$ term because noise is over I and Q samples, whereas we are considering only the real samples in BPSK.

In an AWGN channel that does not add any gain, the channel gain $h = 1$ always. Now r is Gaussian distributed with mean equal to s and variance $\frac{1}{2}N_0$. Therefore, the probability that a symbol r is received given that $s = s_1$ is transmitted is given by.

$$P(r|s_1) = \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(r-\sqrt{E_b})^2}{N_0}} \tag{3.8}$$

Assume that the demodulation decision is to pick s_1 if $r > 0$ and s_2 otherwise. Let us now calculate the probability of error when symbol s_1 is transmitted, that is, probability that s_1 is sent but is mistaken to be s_2 at the receiver. To compute this probability, we integrate $P(r|s_1)$ from Equation 3.8 for values of r that lead to an incorrect demodulation decision, i.e., $r < 0$.

$$
\begin{aligned}
P(e|s_1) &= \int_{-\infty}^{0} P(r|s_1)dr \\
&= \frac{1}{\sqrt{\pi N_0}} \int_{-\infty}^{0} e^{-\frac{(r-\sqrt{E_b})^2}{N_0}} dr \\
&= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-\sqrt{\frac{2E_b}{N_0}}} e^{-\frac{x^2}{2}} dx \\
&= \frac{1}{\sqrt{2\pi}} \int_{\sqrt{\frac{2E_b}{N_0}}}^{\infty} e^{-\frac{x^2}{2}} dx \\
&= Q(\sqrt{\frac{2E_b}{N_0}}) \\
&= Q(\sqrt{2\gamma_b})
\end{aligned} \tag{3.9}
$$

where Q is the Gaussian error function.

Thus the probability of error of BPSK in an AWGN channel is given by:

$$P_e^{BPSK} = Q(\sqrt{2\gamma_b}) \tag{3.10}$$

Note that if the modulation in use were to be QPSK that uses both the I and Q channels, the distance between adjacent constellation symbols reduces by a factor of $\sqrt{2}$. So probability of error for the same SNR γ_b is now given by

$$P_e^{QPSK} = Q(\sqrt{\gamma_b}) \tag{3.11}$$

That is, the probability of error falls almost exponentially with increasing SNR in an AWGN channel for the common modulation schemes.

Uncoded fading channels. Now, in the case of fading channels, the channel response is not constant and varies with time. For example, in the case of Rayleigh fading, the channel gain h is Rayleigh distributed. For a given fixed h, probability of error $P(e|s_1)$ for a BPSK transmission at a channel gain h (written as P_h for brevity) is as computed before:

$$P_h = Q(\sqrt{2|h|^2\gamma_b}) \tag{3.12}$$

We get the overall probability of error during a transmission by averaging over all possible values of h. In the case of a fast fading channel, the channel response h can potentially vary over the entire distribution during the packet transmission. Therefore, we obtain the probability of error by integrating P_h in Equation 3.12 over all values of h in a Rayleigh distribution.

$$P = \int_0^{\infty} P_h p(h) dh \tag{3.13}$$

where $p(h)$ is the density function of the Rayleigh distribution.

The probability of error after integration can be approximated as follows [50]:

$$P_e^{BPSK} = \frac{1}{4\gamma_b} \tag{3.14}$$

For the QPSK modulation, the equations simplify to:

$$P_e^{QPSK} = \frac{1}{2\gamma_b} \tag{3.15}$$

The functions in equations 3.10, 3.11, 3.14 and 3.15, plotted in Figure 3-1, capture the relationship between the expected channel BER and the SNR per bit of the transmitted signal. Observe that the mapping changes substantially depending on whether the channel changes a lot (i.e., fast fading) or remains the same (i.e., AWGN channel) during the packet transmission. In other words, the mapping between the transmitted SNR per bit and the expected channel BER depends on the coherence time of the channel.

How do practical protocols estimate channel BER in uncoded fading channels? If the SNR is measured once per packet (say, using the preamble), then one must use an environment-specific SNR-BER mapping to obtain the BER from the measured SNR. In the absence of an accurate model of the channel, estimating the SNR once per frame will lead to incorrect estimates of the channel BER in time-varying mobile channels. However, tracking the channel continuously with pilots can give us a better estimate of the channel BER without requiring environment-specific training. For example, let γ_i, $i = 1 \ldots N$, denote the SNRs over each of the OFDM symbols in a packet, measured using pilot OFDM subcarriers as described in Section 3.1.1. Assuming the channel is invariant over the duration of an OFDM symbol (4 μs in 802.11), we can compute the channel BER b_i for bits in symbol i using the SNR-BER mappings of the particular modulation scheme in the AWGN channel. Assuming each symbol contains the same number of data bits, the average BER over the packet is given by $\bar{b} = \frac{1}{N}\sum_i b_i$. Therefore, one can compute the channel BER from fine-grained SNR measurements in uncoded fading channels, even without knowing the exact fading characteristics of the channel.

Coded channels. The problem of BER estimation from SNR is much harder in fading channels with channel coding, even when tracking the channel continuously with pilots, because coding introduces a correlation in the probability of error across the decoded bits. That is, we can no longer compute the probability of error of the bits in symbol i as a function of their SNR γ_i alone. Moreover, because the relationship between the SNR and BER is not linear, the average SNR across all the symbols in a frame (computed as $\bar{\gamma} = \frac{1}{N}\sum_i \gamma_i$) does not accurately capture the average BER over the frame ($\bar{b} = \frac{1}{N}\sum_i b_i$). To see why, let $SNR(b)$ denote the SNR corresponding to a BER of b on the SNR-BER curve. Because the SNR-BER curve is concave, $SNR(\bar{b}) > \bar{\gamma}$. That is, the average SNR computed using pilots *underestimates* the quality of the channel during the transmission; as a result, the BER predicted by this method would be higher than the actual BER of the channel. The error is more prominent if the values of b_i are very different from each other, i.e., in a fast fading channel. That is, the mapping from the average SNR over the entire frame to the average BER also depends on the characteristics of the coded fading channel (like the

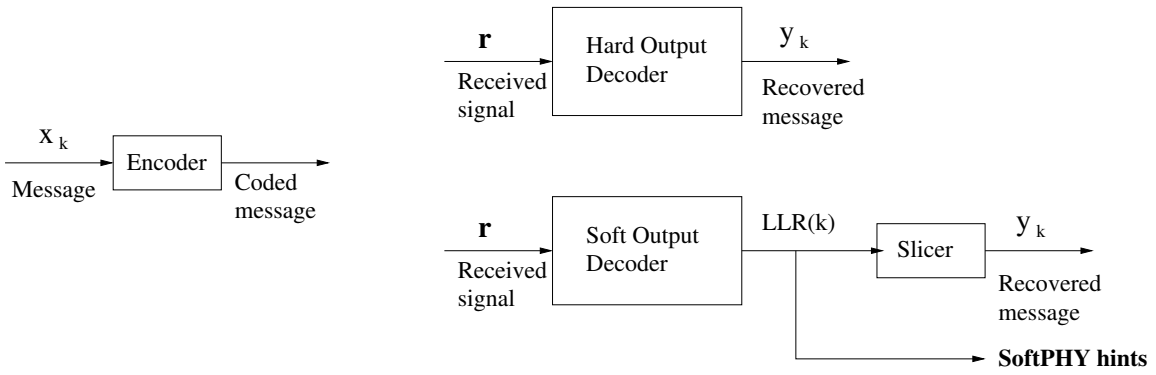


Figure 3-2: An illustration of hard output (e.g., Viterbi) and soft output (e.g., SOVA or BCJR) decoders.

channel coherence time), though to a lesser extent than when measuring the SNR once per frame.

Theoretically, one can still estimate the channel BER from SNR without requiring a channel model: one could simulate the decoder's performance over per-symbol SNR measurements of each frame to measure the bit error rate. For example, recent research on rate adaptation [57] proposes measuring the dispersions of symbols throughput the packet and simulating the performance of decoders at various coding rates within the PHY to estimate the channel quality. However, such a method incurs a high per-packet processing overhead and is impractical to implement in high-speed systems.

As an aside, the relationship between SNR and BER depends on a variety of factors besides the time-variability of the channel, for example, on the interleaving schemes used, the frequency selectivity of the channel, hardware variations, channel estimation errors, and so on [74, 55, 61, 24]. Therefore, any protocol that uses an SNR-BER curve to lookup the channel BER corresponding to a given SNR must recalibrate the curves with changes in the above environmental factors.

3.2 Channel BER Estimation Using SoftPHY Hints

This section describes a method to directly measure channel BER, in a manner that does not depend on the operating environment. We observe that the PHY has more detailed information about wireless channel quality than what is captured by a per-frame SNR. We propose extracting new PHY-layer information about the probability of bit error for each received bit, and use these probabilities to accurately estimate the channel BER.

Recall the receiver tool chain shown in Figure 2-3. The demodulator at the receiver converts the received symbols into bits by mapping the received symbol to the closest valid symbol in the constellation and then to the set of bits that correspond to that valid symbol. Demodulators are of two types. *Hard decision* demodulators produce only the bits that correspond to each received symbol, while *soft decision* demodulators produce the *error vectors* corresponding to the demodulation decision as well. Error vectors are simply vectors representing the distance between the received symbol and the various constellation

symbols. FEC decoders then use the information in these error vectors to guide the decoding process. In finding the most probable input sequence that generated the output sequence of coded bits, input sequences that correspond to output sequences with high error vectors are considered less likely than those with lower error vectors. Soft decision FEC decoding using error vectors from demodulators is more accurate than hard decision decoding and is the default choice in today's wireless systems.

The output of the demodulator is consumed by the FEC decoder to produce the actual message bits. PHYs today use *hard output decoders*, like the popular Viterbi decoder, to recover only the original message bits from the coded bits. Hard output decoders produce only 0s and 1s as their output. Now, a decoder has complete information on which of the decoded bits were received with high error vectors and which weren't, and therefore can estimate how likely it is that a particular output bit is in error. In fact, there exist *soft output decoders* such as the Soft Output Viterbi Algorithm (SOVA) [21] and the BCJR decoder [7] that produce some additional information (e.g., the likelihood of a zero being a zero or a one being a one) in addition to the message bits. In other words, hard output decoders quantize their output into one bit of information while soft output decoders produce a multi-bit output based on the information in the error vectors of coded bits. Soft output decoders are not used in most communication systems because the added hardware complexity to generate soft output is not justified when the fine-grained soft output is not used by the higher layers. However, we observe that by exporting this soft output to higher layers as SoftPHY hints, one can estimate the wireless channel very accurately at the link layer, leading to the design of better link-layer protocols. This improved wireless performance justifies the added hardware complexity of a soft output decoder.

More formally, suppose x_k, $k = 1 \ldots N$, are the input bits to the encoder at the sender, denoted by the N-dimensional vector $\mathbf{x}$. At the receiver, the demodulator passes up the demodulated bits along with the corresponding error vectors to the error correcting decoder, which then recovers the original message bits from the received coded bits. Let $\mathbf{r}$ denote the received signal input (e.g., demodulated bits and error vectors) to the decoder. Let y_k, $k = 1 \ldots N$, or vector $\mathbf{y}$ denote the output from the decoder at the receiver. Hard output decoders like the Viterbi decoder take some form of the received signal $\mathbf{r}$ as input and produces the vector $\mathbf{y}$ as output, as shown in Figure 3-2. On the other hand, soft output decoders compute the *log-likelihood ratios* (LLRs) for each bit[1]. The LLR of bit k is:

$$\text{LLR}(k) = \log \frac{P(x_k = 1|\mathbf{r})}{P(x_k = 0|\mathbf{r})} \tag{3.16}$$

The LLR of a decoded bit indicates whether it is more likely to be a zero or a one. Therefore, to obtain the output bits y_k from LLR(k), the receiver simply checks if the LLR is greater than zero (indicating the bit is more likely to be 1) or not. This operation is usually referred to as "slicing". A soft output decoder followed by a slicer can recover the decoded bits y_k, as shown in Figure 3-2.

$$y_k = \begin{cases} 1 & : \quad \text{LLR}(k) \geq 0 \\ 0 & : \quad \text{LLR}(k) < 0 \end{cases} \tag{3.17}$$

[1]For the LLRs to be accurate, the channel noise must be Gaussian, a reasonable assumption in practice.

Because LLRs are related to the probability of bit error and indicate the health of the decoding process, we propose using the absolute value of LLRs as SoftPHY hints and exporting them to higher layers. We define s_k, the SoftPHY hint for bit k, as

$$s_k = |LLR(k)| \tag{3.18}$$

We will now see the rationale behind this choice of SoftPHY hints. Define the probability of bit error as

$$p_k = P(x_k \neq y_k|\mathbf{r}) \tag{3.19}$$

Then the SoftPHY hint s_k and p_k are related as follows:

$$\begin{aligned} s_k &= |\text{LLR}(k)| \\ &= \begin{cases} \log \frac{P(x_k=1|\mathbf{r})}{P(x_k=0|\mathbf{r})} & : \quad y_k = 1 \\ \log \frac{P(x_k=0|\mathbf{r})}{P(x_k=1|\mathbf{r})} & : \quad y_k = 0 \end{cases} \\ &= \log \frac{P(x_k = y_k|\mathbf{r})}{P(x_k \neq y_k|\mathbf{r})} \\ &= \log \frac{1-p_k}{p_k} \end{aligned} \tag{3.20}$$

Solving for p_k,

$$p_k = \frac{1}{1+e^{s_k}} \tag{3.21}$$

The average of p_k over all bits k in a frame gives us the average BER of the frame, and hence the BER of the channel during the transmission of the frame. Note that the BER computed in this manner will reflect the true channel BER even if the frame is received without any bit errors. Also note that the above technique captures the channel BER accurately irrespective of the channel coherence time, because the SoftPHY hints implicitly track the channel quality continuously, and capture the impact of channel variations due to fading over the entire packet.

Choice of soft output decoders. Soft output decoders have been extensively studied by the research community, and any soft output decoder that computes LLRs can be used by our approach. Note that the two popular soft output decoders, SOVA and BCJR, are actually quite different in their internal operation. The popular Viterbi decoder, and its soft output counterpart SOVA, are examples of *maximum likelihood* (ML) decoders. That is, the decoder finds the decoded sequence $\mathbf{y}$ that maximizes the probability of the observed signal data, i.e., it finds $\mathbf{y}$ that maximizes $P(\mathbf{r}|\mathbf{y})$. On the other hand, the BCJR decoder is a *maximum aposteriori probability* (MAP) decoder which maximizes the aposteriori probability of each decoded bit, i.e., it maximizes $P(y_k|\mathbf{r})$ for each of the decoded bits y_k. To better understand the difference between MAP and ML decoders, consider the following conditional probability relationship based on the Bayes equation.

$$P(\mathbf{y}|\mathbf{r}) = \frac{P(\mathbf{r}|\mathbf{y}) \cdot P(\mathbf{y})}{P(\mathbf{r})} \tag{3.22}$$

MAP decoders find input sequences $\mathbf{y}$ that maximize the LHS of Equation 3.22, whereas ML decoders find sequences that maximize the numerator of the RHS. Now, as long as input bits "0" and "1" are equally likely, the term $P(\mathbf{y})$ is going to be the same in both algorithms, and so both algorithms make the same decoding decisions and recover the same message. However, MAP decoders such as BCJR explicitly compute and maximize aposteriori probabilities. As a result, LLRs are natural byproducts of a MAP decoder. ML decoders such as SOVA do not explicitly compute aposteriori probabilities; the LLRs produced by these algorithms are only approximations that are derived from the state maintained during ML decoding. However, our experimental results using software radios show that the difference in LLRs computed by the BCJR and SOVA decoders is too small to observe in practice and either decoder would work fine to compute SoftPHY hints.

3.2.1 Comparison with SNR

We observe that the error vectors that are inputs to the soft output decoder capture the distortion to the received symbols due to channel noise, and are related to the SNR of the channel. Therefore, our technique of estimating BER from SoftPHY hints amounts to tracking the channel SNR continuously, factoring in the impact of channel coding, and computing the channel BER after FEC decoding. However, there is one significant difference. As we explain in Section 3.1.2, coding introduces a correlation in the error probabilities of decoded bits. Therefore, one requires a knowledge of some propagation channel characteristics (like channel coherence time) to estimate the BER of coded fast-fading channels using SNR measurements. Unlike BER estimation using SNR, SoftPHY hints can be used to estimate the channel BER from the received symbols without requiring a knowledge of the channel fading characteristics.

Note that SNR characterizes the channel quality independent of modulation and coding, while SoftPHY hints capture the probability of error after FEC decoding. Therefore, the SNR of a channel conveys more information about the channel than SoftPHY hints: SNR measurements can be used to predict the channel BER at any modulation and coding, while SoftPHY hints can compute the channel BER only at the modulation and coding of the received frame. However, when the end-goal is measuring higher-layer throughput, it is more convenient to export SoftPHY hints from the PHY instead of SNR measurements, because higher-layer protocols running atop the SoftPHY interface can compute application throughput using the BER estimate from SoftPHY hints across all operating environments.

Another tradeoff between SNR and SoftPHY hints is one of overhead. SoftPHY hints do not incur the additional communication overhead of pilot symbols to track the channel, and can work with just the received payload to estimate channel BER. However, SoftPHY hints do incur a higher overhead in terms of hardware for computation, because soft output decoders consume more hardware resources than hard output decoders. In comparison, estimating SNR does not require adding new modules to existing PHY designs, and adds very little overhead in terms of computational complexity.

3.2.2 Applicability

While commonly used to decode convolutional codes, the SOVA and BCJR algorithms can be extended to decode any linear convolutional or block error correcting code. As such, our method of computing BER can be used in any communication system that uses such codes, examples of which include the popular 802.11a/b/g and Zigbee.

Our method also applies to recent OFDM physical layers such as 802.11n that are based on the new Multiple Input Multiple Output (MIMO) technology. MIMO systems have multiple transmit and receive antennas, which provide multiple paths with different channel losses between the transmitter and receiver. These multiple transmission paths provide *spatial diversity* between the transmitter and receiver. The PHY in a MIMO system can use the multiple antennas in two ways: it can transmit multiple independent data streams through the different antennas and scale the throughput in proportion to the number of antennas, or can harness spatial diversity by transmitting the same data stream along different antennas and recovering the data more reliably. Note that the presence of MIMO does not impact the SoftPHY hint computation because the complexity in MIMO arises in manipulating and combining the analog signal from multiple antennas, while the FEC decoder down the pipeline that operates on the coded bit streams does not have to be concerned with these complications.

Our method can also be extended to advanced communication systems such as WiMax, Digital TV, satellite communications, and cellular LTE, that use concatenated codes for error correction. Concatenated codes involve using two block or convolutional codes either in parallel or serially to perform two stages of error correction, examples of which include the popular Turbo codes [8]. Such codes have been shown to achieve performances close to Shannon's channel capacity. Concatenated codes use two separate decoders for the inner and outer codes. In current PHYs, the inner decoder is usually a soft output decoder because the LLRs from the inner code serve as soft inputs to the outer decoder and improve its performance. However, the outer decoders typically produce hard decisions. By using soft output decoders to decode the outer code as well, one can obtain LLRs and compute SoftPHY hints in such physical layers.

3.2.3 Previous Work on SoftPHY Hints

Earlier research [26, 27] proposed using a different design of SoftPHY hints to identify likely errored bits in a packet reception, in order to selectively retransmit only those bits. These SoftPHY hints were developed in the context of the Zigbee communication scheme that uses a block error correcting code. Recall that decoding block codes involves matching a received codeword to the closest valid codeword. The authors propose deriving the SoftPHY hints from the Hamming distance between the received codeword and the closest valid codeword, with the intuition that the higher this distance, the lower the confidence that this codeword was decoded correctly. One can then guess that all symbols with a Hamming distance above a certain threshold likely have bit errors. While these SoftPHY hints are suitable if the primary aim is to identify likely errored bits, they do not map in any easy way to the underlying channel BER. If applications such as bit rate adaptation wish to know the channel BER in a probabilistic sense even if the received packet has no bit errors,

Modulation	Code Rate	802.11 Rate	Implemented?
BPSK	1/2	6 Mbps	Yes
BPSK	3/4	9 Mbps	Yes
QPSK	1/2	12 Mbps	Yes
QPSK	3/4	18 Mbps	Yes
QAM16	1/2	24 Mbps	Yes
QAM16	3/4	36 Mbps	Yes
QAM64	1/2	48 Mbps	No
QAM64	2/3	54 Mbps	No

Table 3.1: Combinations of modulations and coding rates used in 802.11, the raw throughput achieved over a 20 MHz channel, and their implementation status in our prototype.

then hints that just identify likely errored bits are not quite useful. Moreover, the Hamming distance approach to computing SoftPHY hints does not generalize to communication systems that do not use a block code. We show in our evaluation that the LLR-based SoftPHY hints that we propose are useful both to estimate the probabilistic channel BER as well as identify likely errored bits in a packet. As a result, our scheme of computing SoftPHY hints completely supersedes the earlier approach.

3.3 Implementation

In this section, we describe two prototype implementations of a physical layer that computes per-bit SoftPHY hints, one built on software radios and the other on an FPGA platform. While our software implementation enables us to perform a wide range of experiments with easily available software radios, our FPGA implementation proves that the computation of SoftPHY hints is feasible in real hardware.

In both implementations, the PHY exports the log-likelihood ratios and not the probability of bit error directly. While both choices would have worked fine in a software implementation, exporting LLRs is easier in a hardware implementation with fixed point arithmetic. This is because LLRs are logarithmic and therefore have a smaller range of variation than BERs.

3.3.1 Computing SoftPHY Hints in Software

We built a SoftPHY prototype by modifying an 802.11a/g-like PHY running on the popular GNURadio codebase [2] with the USRP RF front-end. The GNURadio software was based off the OFDM codebase developed by Anastasopoulos and Tom Rondeau. It was subsequently modified by Kyle Jamieson to use a soft output decoder, and later refined by us to compute the SoftPHY hints described in this dissertation.

Our prototype closely follows the basic PHY dataflow illustrated in Figure 2-3. At the transmitter, incoming data passes through a standard rate-1/2 convolutional encoder, after which it is *punctured* to produce coded bit streams at varying rates. The punctured bits

are then mapped to OFDM subcarriers, using either BPSK, QPSK, QAM16, or QAM64 modulations. The combinations of modulations and coding rates used in 802.11a/g, the corresponding raw 802.11 throughput on a 20 MHz channel, and the implementation status in our prototype are shown in Table 3.1. The link-layer header is always transmitted at the lowest base rate, unlike 802.11. The decoding process at the receiver first demodulates the received data, and then decodes the error correcting code using the Viterbi decoder to produce output bits.

In order to compute SoftPHY hints, we replace the Viterbi decoder with a SOVA or BCJR decoder. The decoders take as input from the demodulator the Euclidean distance between the received symbols and the valid constellation symbols. We found that replacing the Viterbi decoder by a soft output decoder adds negligible overhead in terms of both receiver complexity and per-packet processing cost in our software implementation. We also found that both the SOVA decoder and the BCJR decoder computed identical LLRs in our experiments (Section 3.4).

SNR computation. Our prototype also computes an SNR estimate from the packet preamble at the start of each received frame using the popular Schmidl-Cox method [56], and an average SNR over the entire packet using pilot subcarriers, as described in Section 3.1.1. Our experiments evaluate the accuracy of BER estimation from these two types of SNR estimates, in addition to the accuracy from SoftPHY hints.

Fading channel simulator. We implement a Rayleigh fading channel simulator in GNU-Radio using a Jakes simulator model [75]. We use the channel simulator to connect the software radio sender and receiver blocks in a local loopback configuration. The simulator takes, among other things, the coherence time of the channel as a parameter, and simulates channel fades in accordance with the specified coherence time. Therefore we use the simulator to test our implementation in a variety of channel conditions like fast fading due to train-speed mobility that are hard to achieve in the lab environment.

3.3.2 Computing SoftPHY Hints in Hardware

We implemented a SoftPHY-capable PHY using Airblue, an FPGA-based platform for developing cross-layer wireless protocols. Airblue is implemented on a custom hardware platform built around an Altera Cyclone III FPGA, shown in Figure 3-3. The FPGA is connected to a 2.4 GHz Radio Frequency (RF) front-end capable of 20 MHz and 40 MHz baseband modulation. The FPGA communicates with a host computer via a high-speed USB interface. The FPGA software consists of an 802.11a/g-like OFDM baseband PHY as shown in Figure 3-4 and a simple contention MAC, written in the Bluespec [1] hardware programming language and compiled to run on the FPGA. Most of the baseband PHY processes OFDM symbols at a clock speed of 25 MHz, except for the Viterbi decoder that is more computationally intensive and therefore needs to run at a higher clock speed of 40 MHz.

To obtain SoftPHY hints for every bit, we replaced the Viterbi decoder in the receiver pipeline with the BCJR soft output decoder, as shown in Figure 3-4. We use the BCJR

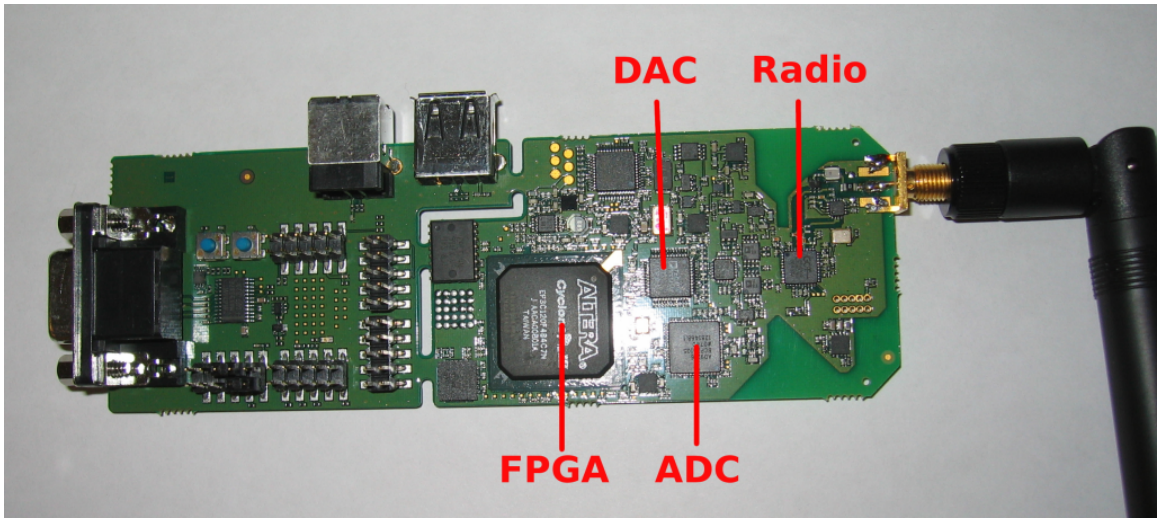


Figure 3-3: The hardware used in Airblue, an FPGA-based platform used to develop a SoftPHY-capable PHY.

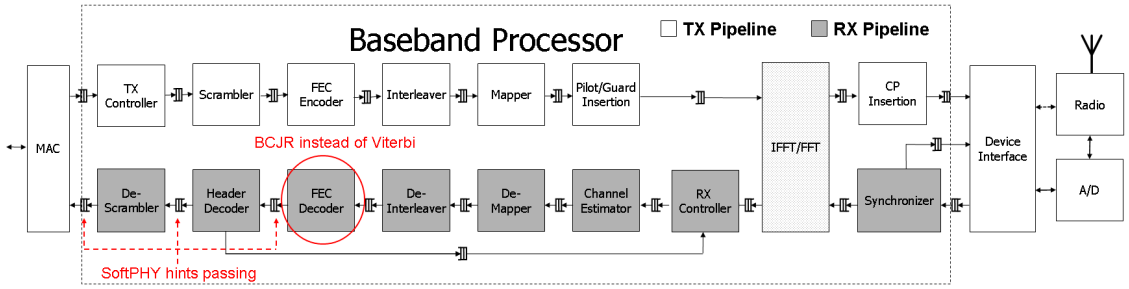


Figure 3-4: OFDM baseband data flow in our FPGA-based PHY on the Airblue platform, and modifications to the pipeline to implement SoftPHY hints.

algorithm instead of the SOVA decoder because the BCJR decoder is commonly used in the hardware community (in the design of decoders for concatenated codes) and has a readily available pipelined design. On the other hand, it was much harder to achieve pipeline parallelism with SOVA, as a result of which our SOVA decoder did not meet the required timing constraints.

It has been shown that the complexity of the BCJR algorithm is at least three times that of the (hard output) Viterbi decoder [53]. As a result, the BCJR decoder requires more clock cycles to compute each output bit when compared to the Viterbi decoder. We define the *receiver pipeline processing latency* as the time between when the transmitter PHY sends out a PHY symbol over the air and the receiver PHY decodes the last bit of that symbol and passes it to the link layer. A dominant part of this latency is the time taken to decode a symbol at the receiver, because air propagation delays are usually negligible at short distances. While our original PHY with the Viterbi decoder had a processing latency of 9.72 μs, the new SoftPHY-capable PHY with the BCJR decoder has a processing latency of 15.52 μs. That is, each bit requires an additional 5.8 μs to be decoded with SoftPHY

Mode	Bandwidth	Tones	Pilot Tones	OFDM Symbol Time
Long range	500 KHz	1024	16	2.6 ms
Short range	4 MHz	512	16	160 μs
Simulation	20 MHz	128	8	8 μs

Table 3.2: Modes of operation of our software radio OFDM prototype. Also shown are the RF bandwidth sampled, number of subcarriers in each OFDM symbol, the number of pilot tones in each OFDM symbol, and OFDM symbol time in each mode. The cyclic prefix length is one-fourth the OFDM subcarrier length in all the modes.

hints.

However, this increase in processing latency does not effect the throughput or the feasibility of our prototype due to the pipelining of the various PHY operations. The pipelined PHY implementation can still decode one PHY symbol every 3.2 μs (the 802.11 OFDM standard specifies that one symbol must be transmitted and received every 4 μs). Therefore, the increased processing latency only increases the time taken to decode the last symbol by 5.8 μs, and reduces the slack time available to transmit a link-layer ACK frame immediately after the transmission. We will now show that our implementation can meet 802.11 timing requirements for sending the link-layer ACK.

If a frame is received correctly, the 802.11 standard specifies that the receiver must wait for a duration equal to the Short Inter Frame Spacing (SIFS) after the completion of a transmission, and then send the ACK frame. A receiver requires one CSMA slot time to detect that the medium has become idle and the transmission has ended. For the 802.11a PHY, the slot time is 9 μs and the SIFS interval is 16 μs. Therefore, the receiver PHY must be able to transmit an ACK within 9 + 16 = 25 μs after the completion of a transmission. Now, the PHY finishes decoding the packet 15.52 μs after the transmission completes (as mentioned earlier). The module that checks the CRC of the frame requires an additional 1.32 μs to check if the frame was received correctly or not. The link layer requires under a microsecond to initiate an ACK transmission to the PHY. Thus, in the case of a correct reception, the link layer is ready to issue an ACK frame in 15.52 + 1.32 = 16.84 μs, well within the 25 μs timeout interval. Even if the exact numbers of the SIFS duration and slot time are different across different standards, one can see that even our unoptimized implementation of a SoftPHY-capable PHY can comfortably meet timing requirements. Moreover, because the decoded bits and SoftPHY hints are available at the link layer well in advance before an ACK frame transmission begins, one can embed the BER feedback computed from SoftPHY hints into the ACK frame comfortably. This BER feedback can then be used by link-layer protocols that require feedback on channel quality. The next two chapters of this dissertation present two such protocols, SoftRate and SoftCMAP, that rely on this feedback to perform bit rate adaptation and channel access respectively.

To export the SoftPHY hints along with the data in the hardware implementation, we extend the data types of the interfaces between the modules downstream to the decoder (the *Header Decoder* and *Descrambler* modules, as shown in Figure 3-4) to hold a 9-bit SoftPHY hint in addition to the data bit. Exporting the hints to the link layer did not further change the processing latency of the receiver pipeline.

Experiment	Used in	Method
Static	§3.4.2	Six static sender-receiver pairs operating in the long range mode were used. Each sender transmitted 100 960-byte packets each at 20 different sender transmit powers and 6 different bit rates.
Walking	§3.4.3, §4.4.3	One sender transmitting in short range mode was moved at walking speed away from the receiver in 10 experimental runs of 10 seconds each. A total of 4,000 packets per bit rate were transmitted.
Simulation	§3.4.3, §4.4.4	A sender and receiver were connected by our GNU Radio fading channel simulator. The Doppler spread of the channel was varied from 40 Hz to 4 KHz. One hundred packets each were transmitted at 20 different transmit powers of the sender at each of the Doppler spread values.
Static (interference)	§3.4.4, §4.4.2	A sender and interferer transmitted packets simultaneously to a receiver in the long range mode. A random jitter of around one packet-time was added between both transmissions. One hundred packets each were transmitted at 20 different sender transmit powers, five different interferer transmit powers and six different bit rates.
Static (short range)	§4.4.5	Single static sender transmitted packets in short range mode in 10 experimental runs of 10 seconds each. A total of 4,000 packets were transmitted at each of the bit rates across all the runs.

Table 3.3: A summary of the experiments with the software radio prototype used to evaluate SoftPHY hints.

3.4 Evaluation

In this section, we analyze packet traces from experiments and simulations on a SoftPHY-capable PHY to measure the accuracy of BER estimation using SoftPHY hints. Our main findings are summarized below.

1. SoftPHY hints can accurately estimate channel BER across a wide variety of wireless propagation channels, and in the presence and absence of interference.
2. BER estimation with SoftPHY hints has a lower variance when compared to existing ways of estimating BER from measured SNR.
3. SoftPHY hints can not only estimate the probabilistic channel BER but can also identify the locations of the bit errors.

3.4.1 Method

We use a combination of live experiments and controlled simulations using our software radio and hardware OFDM prototypes to collect extensive packet traces with SoftPHY hints. In addition to raw packets, the traces also record the SoftPHY hints and SNR measurements of received frames.

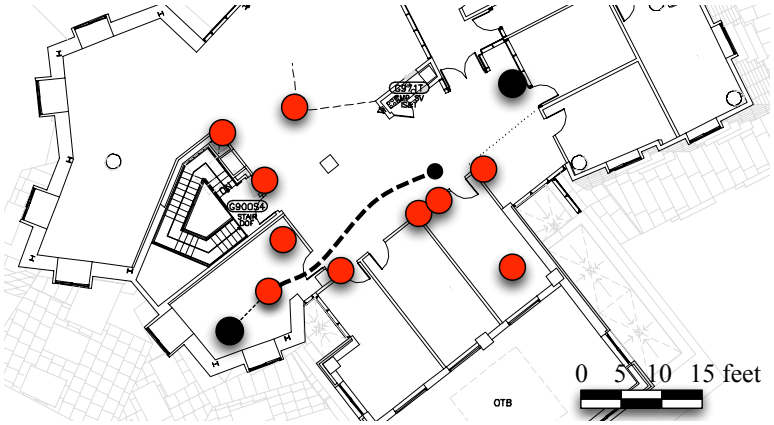


Figure 3-5: Evaluation testbed of software radio nodes: light (red) shaded nodes are senders; black nodes are receivers. The thick dashed line shows the approximate path of the sender in mobility experiments.

Trace collection with the software radio prototype. We run our software radio experiments in two modes. In the *long range* mode, the USRP samples a smaller RF bandwidth in the 2.4 GHz band than in the *short range* mode. Because of sampling error, the latter results in signals of lower fidelity from the USRP, resulting in it being unusable over a few links in our testbed. However, a smaller RF bandwidth in the long range mode also leads to typical frame durations of tens of milliseconds. As a result, only experiments in static topologies that see little variation at that timescale were run in the long range mode. In contrast, the short range mode results in frames that last less than a millisecond, making it suitable to run mobility experiments in fading channels that change on shorter timescales. Experiments using our fading channel simulator (instead of the real RF channel) were not limited by the RF front-end; such experiments were run over the normal 20 MHz band with 802.11-like frame durations. We summarize these modes of operation in Table 3.2. Note that all the modes use more subcarriers than used in commodity 802.11 cards today (i.e., 48) because a higher number of subcarriers enables better physical layer synchronization and channel estimation.

We run a variety of live experiments in static and mobile configurations on the testbed shown in Figure 3-5 by varying the locations of the senders and receivers, and the transmit power and bit rate. We also run controlled simulation experiments with our fading channel simulator by varying the Doppler spread parameter of the fading channel from 40 Hz to 4 KHz. Recall from Chapter 2 that if the Doppler spread in frequency due to mobility is f, then the coherence time of the channel is roughly $\frac{0.4}{f}$ [65]. Therefore our channel simulator traces correspond to channel coherence times between 10 ms and 100 μs, and capture a variety of channel conditions ranging from movement at walking speed in indoor environments (coherence time close to 30 ms) to movement at train speeds (coherence time close to 100 μs). Table 3.3 elaborates on the various experiments and traces used in the evaluation. These packet traces are also used in subsequent chapters for evaluation of other protocols based on SoftPHY hints.

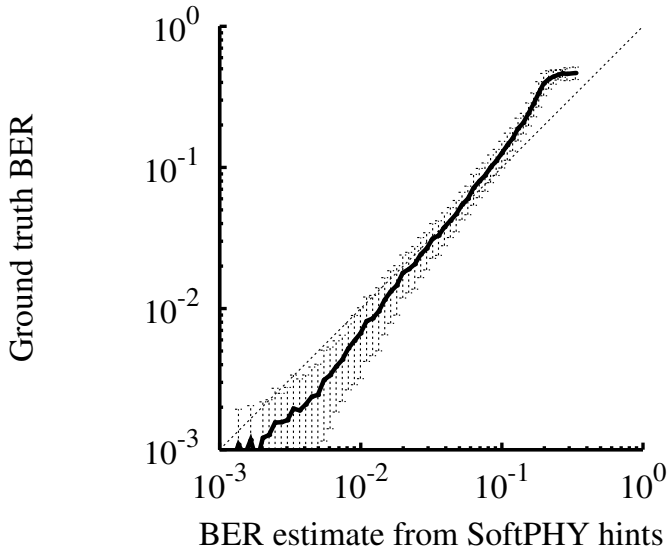


Figure 3-6: Per-frame average BER estimated by SoftPHY hints vs. the actual BER of the frame from experiments in a static channel.

Trace collection with Airblue prototype. While the easy availability of the mature GNURadio-based software radios enables us to run a large number of live experiments involving multiple radios, the RF front-end in the Airblue platform is still being debugged and is not suitable for running over-the-air experiments. Instead, we run a cycle-accurate hardware simulation of the hardware transmitter and receiver that are connected over a simulated AWGN channel. We transmit 1000 4000-byte packets between a transmitter and a receiver, with different noise conditions being simulated in the AWGN channel simulator, and collect the packet traces and SoftPHY hints from the hardware simulation. While not extensive, these traces demonstrate the feasibility of our hardware implementation.

3.4.2 BER Estimation in Static Channels

We first analyze data from the static experiment described in Table 3.3. For each frame in the trace, we compute the probability of error p_k for each bit k from the SoftPHY hint s_k using Equation 3.21. Then we average p_k over the frame to compute a per-frame average BER. Separately, we determine the frame's ground truth BER by checking the received bits against the known payload. We aggregate the results across different transmit powers, sender-receiver pairs, and bit rates in the trace. We bin the BER estimate data in fixed-sized bins of 0.1 units in the SoftPHY metric (roughly logarithmically-sized bins of the estimated BER). Figure 3-6 plots the true BER of the frame against the BER estimated from SoftPHY hints; error bars in this and subsequent figures indicate one standard deviation about the mean. We see from the figure that the SoftPHY-based BER is an good estimator of true BER.

The preceding experiment tests BER estimation frame by frame. But it is hard to reliably observe BERs below 10^{-3} in one 960-byte frame. We therefore aggregate *all* the frames associated with a SoftPHY-based BER estimation bin in the above experiment, and

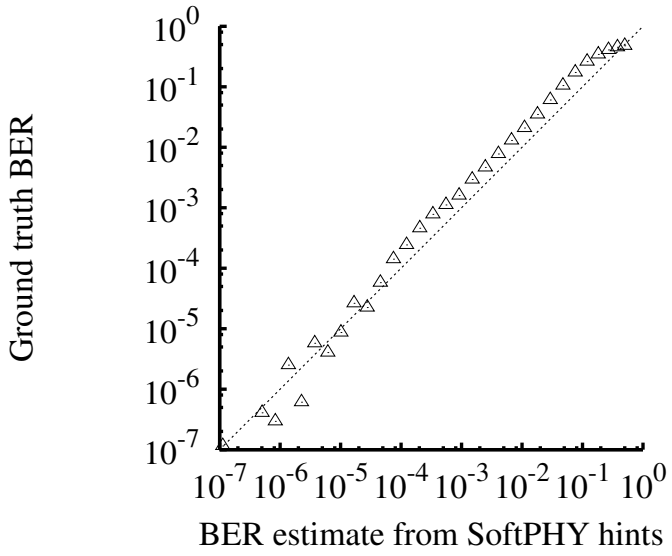


Figure 3-7: Average BER estimated by SoftPHY hints vs. the actual average BER over aggregated bits binned by the estimated BER from experiments in a static channel.

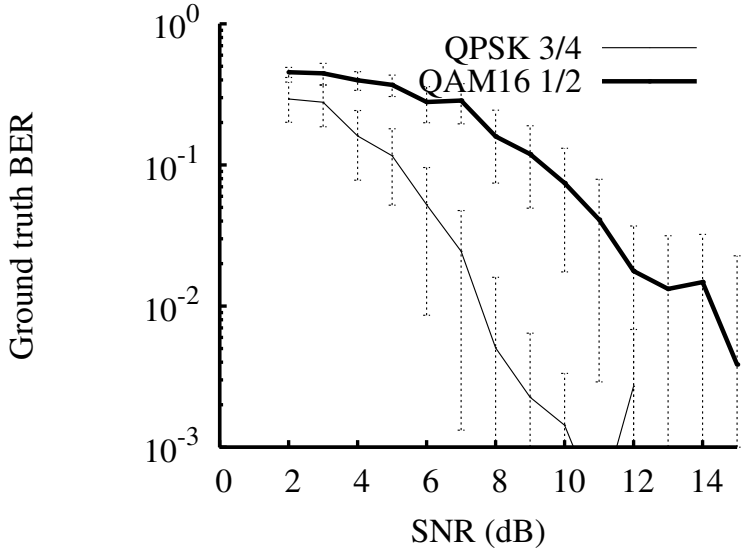


Figure 3-8: SNR of a frame computed from the preamble vs. the actual BER of the frame for the QPSK 3/4 and QAM16 1/2 bit rates from experiments in a static channel.

compute the average BER over the aggregated bits (Figure 3-7). We see that SoftPHY hints accurately estimate ground truth BER all the way down to 10^{-7}; the aggregate bits in the graph bins were not sufficient to measure lower BERs.

To analyze SNR-based BER predictions, we separate the trace data by bit rate, because the SNR-BER relationship changes for different modulation and coding schemes (unlike in the case of the SoftPHY-BER relationship). Figure 3-8 shows the ground truth BER plotted against the SNR estimate of the frame computed from the preamble for two bit rates, with

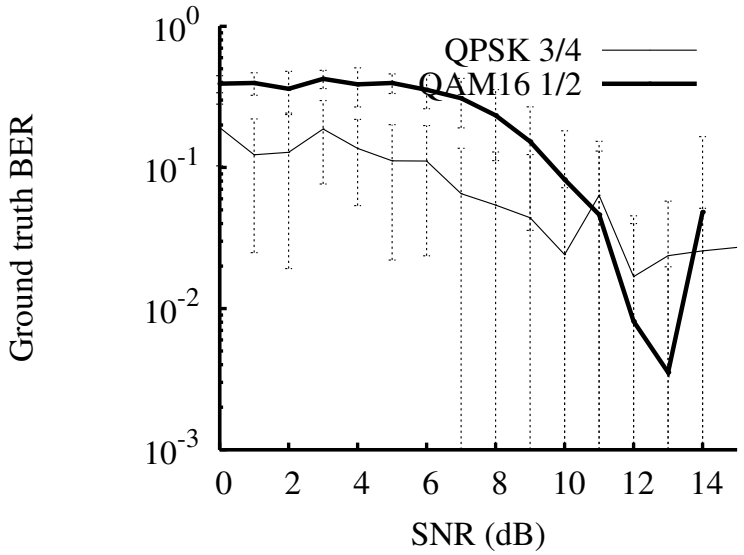


Figure 3-9: Average SNR of a frame computed from pilots vs. the actual BER of the frame for the QPSK 3/4 and QAM16 1/2 bit rates from experiments in a static channel.

the data binned as described earlier. Figure 3-9 shows the ground truth BER plotted against the average SNR over the entire frame computed using pilots. We see that the error bars are larger with SNR-based BER estimates.

We now quantify and compare the accuracy of BER estimation using SoftPHY hints and SNR. Let $i = 1 \ldots n$ denote the various bins on the x-axis in the above figures. For each bin i, let m_i denote the mean ground truth BER plotted on the y-axis and let d_i denote the standard deviation of the BER plotted on the y-axis. We define a metric called the *normalized deviation* δ as the average ratio of the deviation to the mean across all the bins:

$$\delta = \frac{1}{n} \sum_i \frac{d_i}{m_i} \tag{3.23}$$

We normalize the deviation by the mean value of the BER because a higher value of the deviation, say 0.1, is less significant for BERs around 0.1, but very significant for BER around 10^{-5}. Because the BER varies by many orders of magnitude on the y-axis in the above figures, we normalize the deviation by the mean value of the bin.

The normalized deviation δ for BER estimation from SoftPHY hints (Figure 3-6) is 0.33. The δ for the QPSK 3/4 and QAM16 1/2 bit rates is 0.91 and 0.98 respectively when using SNR from preamble (Figure 3-8), and 1.20 and 1.34 respectively when using SNR from pilots (Figure 3-9). These numbers show us that BER estimation from SoftPHY hints has three times less error than some common ways of estimating BER from SNR values. Also, SNR estimation from pilots, though more convenient in fast varying mobile channels, has a higher margin of error than SNR estimation from the preamble. The likely cause of this observation is that SNR estimation from pilots uses only a handful of sample points per symbol to estimate the SNR as opposed to SNR from the preamble that is computed across all subcarriers in a symbol. To confirm this hypothesis, we conducted an experiment

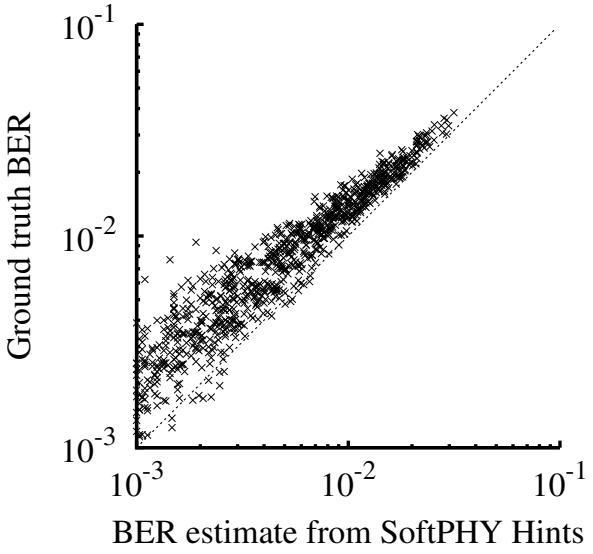


Figure 3-10: Per-frame average BER estimated by SoftPHY hints vs. the actual BER of the frame from simulations with our hardware prototype.

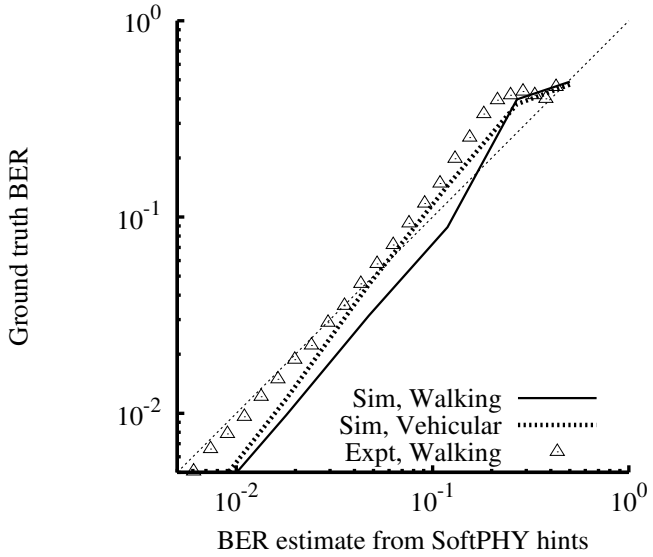


Figure 3-11: Per-frame average BER estimated by SoftPHY hints vs. the actual BER of the frame from experiments (points) and simulations (curves) in a mobile channel.

where a known payload was transmitted and all the data bits were used as pilots to compute the SNR. The normalized deviation for the two bit rates in the experiment was found to be 0.73 and 0.75 respectively.

Figure 3-10 plots the ground truth BER of the frame against the BER estimated from SoftPHY hints for the traces collected from simulations with our hardware prototype. We see from the figure that the SoftPHY-capable PHY implemented in hardware performs similar to the one in software radios, and can accurately estimate channel BER.

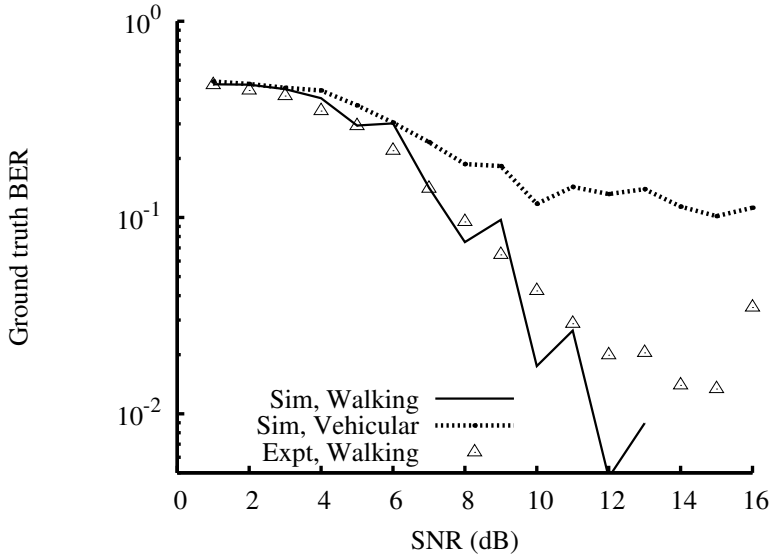


Figure 3-12: SNR of a frame computed from the preamble vs. the actual BER of the frame for the QAM16 1/2 bit rate from experiments (points) and simulations (curves) in a mobile channel.

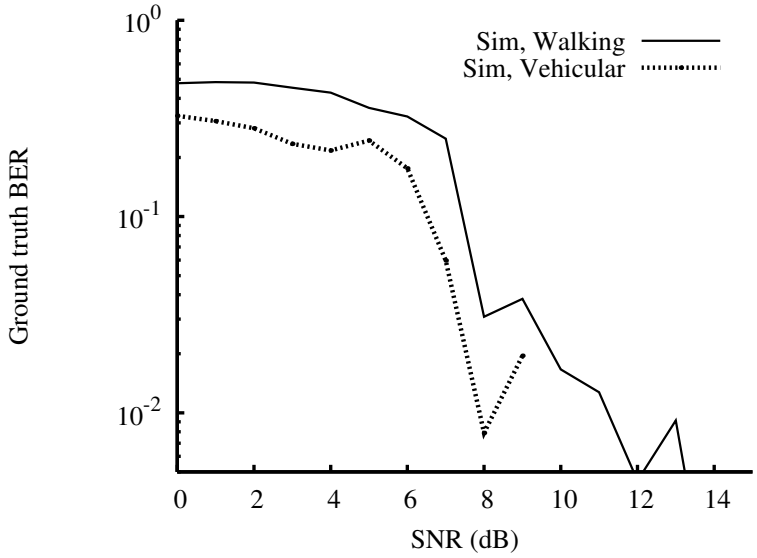


Figure 3-13: Average SNR of a frame computed from pilots vs. the actual BER of the frame for the QAM16 1/2 bit rate from simulations in a mobile channel.

3.4.3 BER Estimation in Mobile Channels

We now show that SoftPHY hints reliably estimate BER even in mobile fading channels with widely varying channel coherence times. This section uses data from the walking and simulation traces of Table 3.3. For each dataset, we bin the data by SoftPHY-estimated BER, and compute the mean ground truth BER in each bin. Figure 3-11 shows the results,

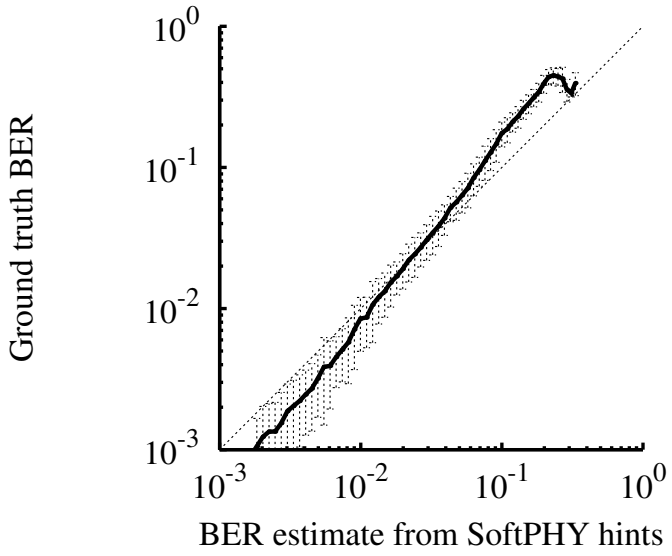


Figure 3-14: Per-frame average BER estimated by SoftPHY hints vs. the actual BER of the frame from experiments with two concurrent transmissions.

with the two curves corresponding to simulation traces at walking (Doppler spread 40 Hz) and vehicular speeds (Doppler spread 400 Hz), and the points in the figure corresponding to experimental data from the walking traces. The corresponding SNR-BER curves at the QAM16 1/2 rate are shown in Figure 3-12 for SNR computed from the preamble. The SNR-BER curves when the average SNR is measured from pilots for walking and vehicular speeds are shown in Figure 3-13.

From the figures, we see that the SoftPHY-based BER estimate is not sensitive to mobility speed. SoftPHY hints reflect the increasing number of deep fades in the body of the frame as channel coherence time decreases, and therefore estimate BER across all wireless propagation environments accurately. In the case of SNR measured from the preamble, the SNR measured at the start of the frame does not capture the variation of SNR that happens over the body of the frame in fading channels. As a result, for the same measured SNR in the preamble, a higher BER is observed in channels that vary more. The fact that the SNR-BER relationship changes with channel coherence time is well-known [50] and has also been observed experimentally by Camp and Knightly [13]. The SNR-BER curve obtained by using the average SNR from pilots is much less sensitive to the environment than that obtained using the SNR from the preamble. We find that averaging the SNR using pilots slightly underestimates the actual SNR (for reasons described in Section 3.1.2), causing the SNR-BER curve to shift lower by a small amount.

3.4.4 BER Estimation with Concurrent Transmissions

Certain protocols (see Chapter 5) require estimating the channel BER not just during a single transmission but also in the presence of multiple concurrent transmissions. Consider a transmission from a sender to the receiver, during which another transmission from an interferer starts. The problem now is to estimate the BER over the entire frame, so as to

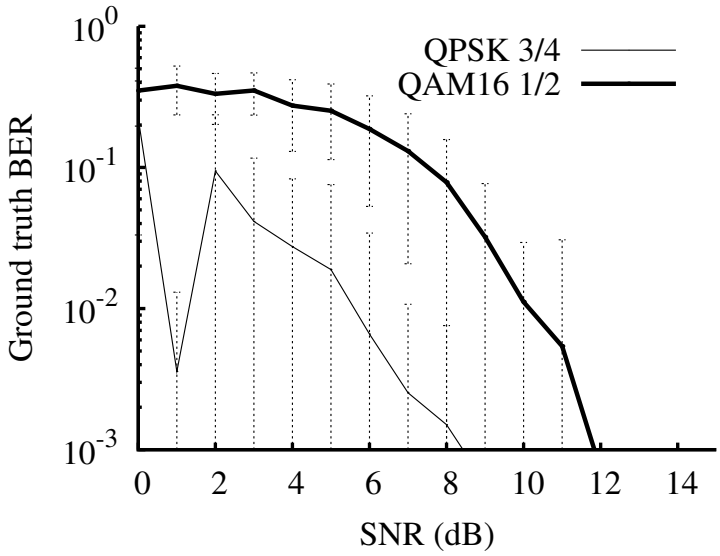


Figure 3-15: Average SNR of a frame computed from pilots vs. the actual BER of the frame for the QPSK 3/4 and QAM16 1/2 bit rates from experiments with two concurrent transmissions.

factor in the effect of the interference. Clearly, predicting BER from the SNR measured at the start of the packet does not reflect the effect of the interference that starts later. On the other hand, BER computed from SoftPHY hints or average SNR over a frame reflects the effect of the concurrent transmission.

We analyze packets in the static interference trace of Table 3.3. We analyzed all the packets detected at the receiver and estimated per-frame BER using SoftPHY hints. Figure 3-14 shows the per-frame BER estimated from SoftPHY hints plotted against the actual BER of the frame. On the other hand, Figure 3-15 shows the average frame SNR computed from pilots plotted against the actual frame BER. We see from the figure that SoftPHY hints can be used to accurately measure the channel BER in the presence of multiple transmissions. We also see that BER estimation from SNR has a higher variance than that using SoftPHY hints, as we observed in the previous experiments.

3.4.5 Identifying Bit Errors with SoftPHY Hints

In addition to being useful in estimating the channel BER in a probabilistic sense, SoftPHY hints can also be used to identify the exact bits in a received frame that are likely to be in error, for applications like error recovery [27] and symbol-level opportunistic routing [33]. We separate out the correct and incorrect bits from all the packets in the static trace of Table 3.3 and sort them by their SoftPHY hints. Figure 3-16 shows the CDF of SoftPHY hints of correct and incorrect bits. From the graph, one can easily see a clear separation in the values of the confidences for correct and incorrect bits. Therefore, one could use a threshold on the SoftPHY hints and guess that all the bits with hints below the threshold are incorrect. Of course, this process inevitably leads to some false positives (mislabeling correct bits as bit errors) and false negatives (failing to identify actual bit errors). The

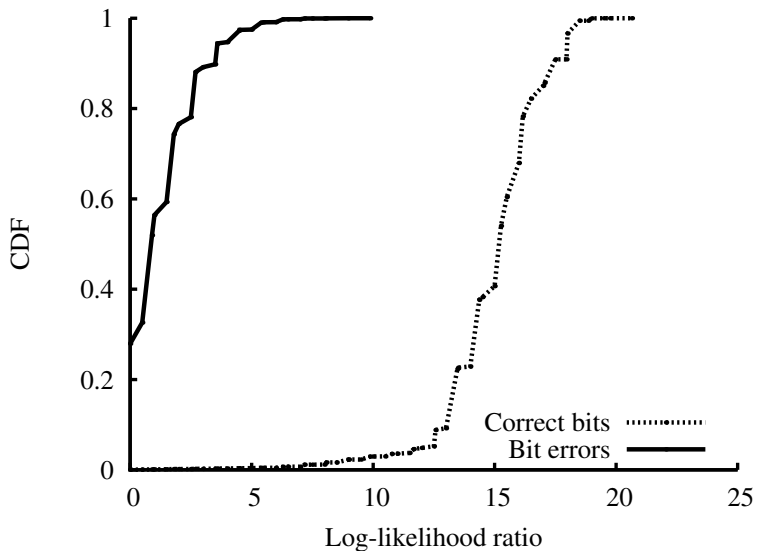


Figure 3-16: CDF of the SoftPHY hints of bits decoded correctly and incorrectly.

choice of the threshold often depends on the requirements of the higher-layer protocol with respect to false positives and false negatives. For example, if we choose a SoftPHY value of 5.5 as the threshold for identifying bit errors, then we see from the figure that we mislabel about 1% of incorrect bits as correct (false negatives) and 0.5% of correct bits as incorrect (false positives). If we use a threshold of 9.9, we identify all incorrect bits, and only label 3% of correct bits as incorrect.

3.5 Chapter Summary

Estimating channel BER is a useful primitive for many link-layer protocols, and is the first step in estimating higher-layer throughput. Link-layer protocols today rely on proxies for channel BER such as SNR measurements and frame loss rates. While frame loss rate measurements require multiple frame transmissions to converge to a meaningful estimate, one requires a model of the propagation channel to accurately compute channel BER from SNR measurements. This chapter proposes a new method for directly estimating wireless channel BER using the per-bit confidences or SoftPHY hints exported by the physical layer. We propose using the log-likelihood ratios produced by soft output decoders in the PHY as SoftPHY hints; this design works for any PHY that uses error correction. These SoftPHY hints can then be used to estimate the channel BER using a single frame reception and without knowing the transmitted payload. This estimation of BER from SoftPHY hints works across all modulation and coding schemes, and does not depend on the channel coherence time and other fading characteristics. We implement SoftPHY-capable PHY prototypes using GNURadio-based software radios and an FPGA-based hardware platform. Our experiments with these prototypes show that SoftPHY hints can accurately estimate channel BER across a wide range of channel conditions.

Chapter 4

Bit Rate Adaptation Using SoftPHY Hints

This chapter describes the design, implementation, and evaluation of the SoftRate bit rate adaptation protocol. Recall that to improve wireless throughput in a time-varying channel, a wireless transmitter must dynamically adapt its modulation and coding by picking a suitable bit rate to suit channel conditions. The bit rate adaptation protocol makes this choice. To do so, it must address two important questions:

1. What signal (information) should the sender use to select the right bit rate?
2. Over what timescale should this signal be observed?

Let us examine the prior work on bit rate adaptation (Section 4.1) using these two criteria. All prior work broadly uses one of two information signals: *frame receptions* or *signal-to-noise ratio (SNR)*. Frame-based protocols make decisions at the timescale of hundreds of milliseconds and turn out to be too slow to work well in a fast changing mobile channel. On the other hand, SNR-based protocols can make transmit bit rate decisions every one or few frames, but SNR is not a good information signal because the mapping between the SNR and channel BER at a given bit rate may depend on the characteristics of the propagation channel.

Moreover, the information signal used by rate adaptation protocols must also be robust to interference. A bit rate adaptation protocol must not reduce bit rate in response to collisions, because doing so increases the transmit duration of frames and conflicts with other mechanisms (like exponential backoff) that the channel access protocol employs to avoid a collision on the next retry. A frame reception is an example of an information signal that is not robust to interference [71, 52].

The SoftRate bit rate adaptation protocol (Section 4.2) addresses all the above pitfalls of existing protocols. SoftRate uses the interference-free BER estimated from SoftPHY hints at the receiver as its feedback signal, using a heuristic to identify and eliminate the impact of interference on the measured channel BER. By sending this BER estimate in a small feedback frame to the sender, SoftRate adapts the transmit bit rate at the granularity of individual frames, and is highly responsive to rapid channel variations due to mobility.

To evaluate SoftRate, we use traces collected using the SoftPHY-capable physical layer implemented using the GNURadio codebase (see Section 3.3). Because the communication latency from the PHY to the link layer in software radios does not allow for a viable implementation of SoftRate on this software radio platform, we use the PHY-level packet traces collected using our SoftPHY-capable GNURadio prototype and run a trace-driven simulation of the link layer and higher layers using the ns-3 simulator [3]. We implement SoftRate and other rate adaptation protocols in the link layer of ns3 (Section 4.3) and compare their performance in terms of the throughput of a TCP application at the higher layer. Our trace-driven evaluation of TCP over SoftRate (Section 4.4) shows that SoftRate achieves gains of around 20% over SNR-based protocols trained on the operating environment, from 35% to 4× higher throughput than untrained SNR-based protocols, and 2–4× more throughput than state-of-the-art frame-based protocols in mobile fading and interference-dominated channels. Performance gains in our experiments stem from SoftRate's ability to quickly react to rapid channel variations before TCP's end-to-end congestion control mechanism reacts to burst losses, and its resilience to collision-induced losses.

4.1 Related Work

4.1.1 Frame-based Bit Rate Adaptation

Many frame-based rate adaptation schemes have been proposed [31, 37, 34, 47], the most recent ones being RRAA [71] and SampleRate [11]. Frame-based schemes are, by design, less responsive to channel variations than SoftRate because one requires multiple frame receptions to accurately estimate channel state at any bit rate.

Auto Rate Fallback (ARF) [31] is an older bit rate selection protocol used by Lucent WaveLAN-II 802.11 cards. ARF increases the bit rate on ten consecutive successful transmissions at a bit rate and reduces the bit rate on even a single loss. At any bit rate, it uses some fixed number of successful transmissions before attempting the next-higher bit rate. Adaptive ARF (AARF) [37] adapts this parameter by doubling it every time the packet following a rate increase fails. CARA [34] is a variant of ARF that enables RTS/CTS on a frame loss and disables RTS/CTS on a frame success. It incurs significant overhead, and may not converge in the presence of hidden terminals. ONOE [47], a prior bit rate adaptation protocol used in the Atheros driver, uses a credit-based scheme to monitor loss rates at each bit rate, attempting to choose the highest bit rate with less than 50% frame loss rate. These four protocols above have been shown to perform worse than SampleRate [11] because the heuristics used for rate increase and decrease lead to incorrect rate choices in practice.

SampleRate is currently (as of May 2010) used in the Linux 802.11 device driver for Atheros cards. SampleRate maintains a ten-second history of the average time it has taken to transmit a packet at different bit rates, updated every time a packet transmission succeeds or fails (after reaching the maximum retry limit). The time taken to transmit a packet includes the transmission time over the air, the time taken for retransmissions, and the time spent backing off between retransmissions. When a node has a packet to send, the sender picks the bit rate that has the lowest average transmission time. SampleRate also

periodically samples from bit rates other than the current best in order to adapt to changing channel conditions. Every ten packets, SampleRate picks a random bit rate from the rates that have not failed four successive times (to exclude rates with very bad performance) and that have a minimum packet transmission time (i.e., transmission time over the air, without retransmissions or backoff) lower than the current rate's average transmission time.

Robust Rate Adaptation Algorithm (RRAA) uses frame loss information gathered over a short-term estimation window and adapts bit rate more opportunistically than SampleRate. RRAA picks the bit rate that maximizes the rate of delivery of frames based on the frame loss rate computations at various bit rates. Each rate in RRAA is associated with a maximum tolerable loss rate threshold, computed as the loss rate at which the expected throughput of that bit rate equals the expected throughput of the next lower rate. If the frame loss rate at a bit rate, computed over the estimation window, exceeds this threshold, RRAA reduces the bit rate. RRAA also uses a threshold for rate increase, opportunistically increasing the bit rate if the frame loss rate falls below that threshold. The size of the estimation window over which the loss rates are calculated is chosen to be at a granularity finer than the thresholds; this value is typically a few tens of frames in 802.11. RRAA also compares the frame loss statistics both with and without RTS/CTS in order to guess whether each loss is caused by a collision or fading on the channel. It then adaptively enables RTS/CTS more frequently as collision losses increase.

COLLIE [52] makes the observation that collision losses adversely impact the performance of rate adaptation protocols. To address this problem, the COLLIE receiver echoes received frames to the sender. The COLLIE sender analyzes the patterns of bit errors in receptions in order to infer whether an error was due to a collision or a channel loss. COLLIE modifies the ARF frame-based rate adaptation protocol to adapt bit rate on channel losses alone. The COLLIE sender analyzes the PHY's reported received signal strength, the bit error rate, the symbol error rate (based on knowledge of the granularity at which the 802.11 OFDM PHY maps bits to symbols), and errors per symbol (the average number of bits in error among all errored symbols) of the echoed frames. Based on some post-processing of the above metrics, the sender is able to separate collisions from interference with an accuracy of between 60%–95%, resulting in throughput improvements of 20%–60% over the unmodified ARF protocol. However, COLLIE incurs significant overheads due to the feedback of entire received frames from the receiver to the sender. The performance gains of COLLIE over the unmodified ARF protocol, which reduces the bit rate even on a single loss, are unlikely to hold in the case of a smarter rate adaptation protocol.

Finally, other protocols [5, 10] use timing information from the physical layer (such as "channel busy" time from Madwifi or packet interarrival times) to infer and separate out interference losses, but are susceptible to the same inefficiencies as frame-based protocols in general.

4.1.2 SNR-based Bit Rate Adaptation

Because the mapping between SNR and channel BER is easy to obtain in static and slow fading channels, it is conceivable that SNR estimates of received frames can be used to pick the best transmit bit rate that maximizes throughput. RBAR [23] uses the RTS/CTS exchange at the beginning of a packet to estimate and obtain the SNR at the receiver, and

picks the transmit bit rate accordingly. OAR [54] builds on RBAR, opportunistically transmitting back-to-back frames when the channel quality is good. However, in later work [13], the authors show that OAR's assumption of coherence time greater than a packet time does not hold in an outdoor urban environment. CHARM [29] leverages the reciprocity of the wireless channel[1] to estimate average SNR at the receiver using packets overheard from the receiver at the sender, thereby avoiding the overhead of RTS/CTS and enabling implementation on commodity cards. CHARM uses an exponential weighted moving average of the SNR over the past few frames to get an estimate of the channel SNR.

The problem of rate adaptation using SNR becomes more complicated in fast fading channels that occur in outdoor environments with vehicular mobility. Camp and Knightly [13] evaluate a number of SNR-based rate adaptation protocols in outdoor vehicular channels, where the SNR is measured once per frame using the AGC gain at the start of the packet transmission. The authors find that because the SNR-BER relationships change with varying degrees of mobility, SNR-based protocols require in-situ training to perform efficiently across different propagation environments. Our evaluation in Section 4.4.4 confirms this result. While CHARM [29] proposes a mechanism to calibrate its protocol on a slower timescale to handle heterogeneous hardware, their mechanism is not effective against changes in the propagation environment that occur on a faster timescale, say, due to changes in mobility.

The rate adaptation mechanisms in IS-856 CDMA cellular data (1x EVDO) and High Speed Downlink Packet Access (HSDPA) use pilots to track the average SNR over the entire duration of the packet in fast fading channels, reducing the impact of the propagation environment on the SNR estimate. Some researchers have also proposed using pilot subcarriers to measure SNR across different subbands of the transmission frequency band, to adapt bit rate across frequency in frequency-selective channels [16, 51]. In addition to incurring the overhead of pilots, such an SNR estimate is also not robust to interference, as it is not possible to distinguish the reduction in SNR due to interference from that due to a weak signal.

Finally, researchers have observed that it is difficult to accurately measure SNR in current commodity 802.11 systems due to hardware calibration issues and interfering transmissions [74].

4.2 Design

Like most other link-layer bit rate adaptation protocols, SoftRate aims to maximize link-layer throughput. The link-layer throughput achieved at a certain channel BER and bit rate depends on the error recovery mechanism used (e.g., does the link layer retransmit entire frames, or only the bits in error?). Therefore, SoftRate's use of BER as the bit rate adaptation signal has two benefits:

1. *BER is an accurate predictor of performance.* It is a sufficient statistic that predicts the throughput of various error recovery protocols; as a result, SoftRate cleanly inte-

[1]Channel reciprocity states that the propagation characteristics of the wireless channel between the transmitter and receiver are symmetrical.

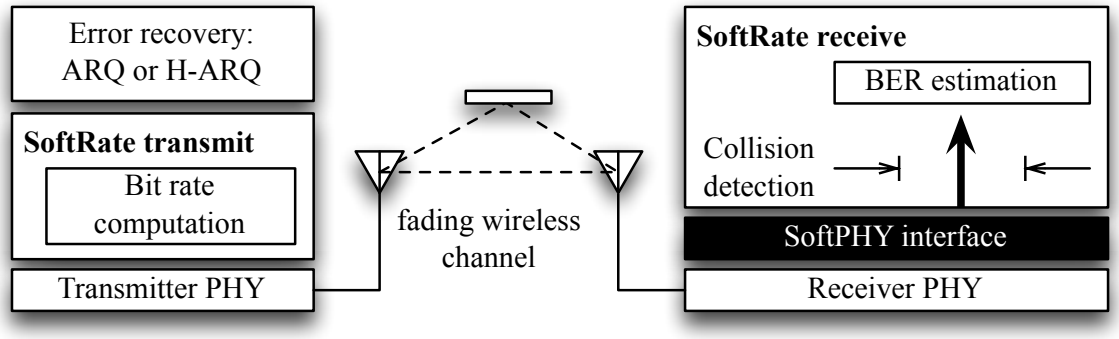


Figure 4-1: A high-level view of the SoftRate system.

grates with many error recovery schemes, as we show later.

2. *BER is responsive.* It can be calculated over short timescales on the order of individual frame transmissions, which allows SoftRate to respond to rapid changes in channel conditions.

The SoftRate protocol works as follows. Receivers in SoftRate use a PHY that is capable of exporting SoftPHY hints. We use the log-likelihood ratios (LLRs) of bits exported by soft output decoders as the SoftPHY hints, as described in Section 3.2. The SoftRate receiver uses these SoftPHY hints to compute the average interference-free BER for each received frame, employing a heuristic to detect and excise those portions of the frame subject to strong interference (Section 4.2.1). The SoftRate receiver then sends the interference-free BER estimate to the sender in a link-layer feedback frame. At the sender's link layer, the SoftRate rate selection algorithm (Section 4.2.2) uses the per-frame BER feedback to pick the best transmit bit rate for the next frame. The algorithm works seamlessly across a wide range of wireless channels (Section 4.2.3).

To ensure reliable delivery of feedback, SoftRate always sends its link-layer feedback frame at the lowest available bit rate in a "reserved" time slot, much like 802.11 link-layer ACKs. The receiver sends feedback whether or not the frame was in error, as long as the frame's preamble and header were decoded correctly. To correctly determine the identities of the sender and receiver even when the frame has an error, link-layer headers are protected with a separate CRC. If the frame has no errors, then the BER feedback is one component of the link-layer ACK. Thus the SoftRate protocol incurs very little extra overhead compared to existing protocols—a CRC in the link-layer header, and a BER measurement in the link-layer ACK. If the sender does not receive any feedback for a frame, the most likely cause is a noisy channel preventing the receiver from even detecting the frame. Therefore SoftRate moves to a lower bit rate if it does not receive feedback for a few consecutive frames.

Figure 4-1 shows the architecture of the SoftRate system. SoftRate operates using only information provided via the layered SoftPHY interface and can inter-operate with any PHY that exports SoftPHY hints; as explained in Section 3.2.2, most existing PHYs qualify.

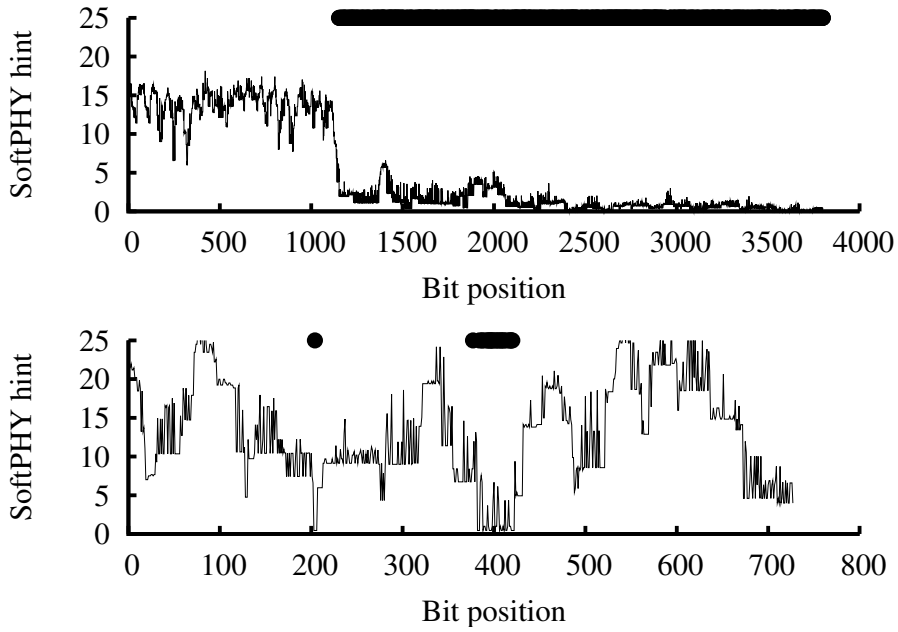


Figure 4-2: Patterns of SoftPHY hints for a frame lost due to a collision *(upper)* and due to channel fading *(lower)*. A circle over a bit-position at the top of the graph indicates a bit error.

4.2.1 Interference Detection

A bit rate adaptation protocol that reduces the transmit bit rate in response to interference losses increases the contention on the channel and exacerbates the interference. However, a responsive rate adaptation protocol that reacts to short-term frame loss rate or BER faces the danger of reacting aggressively to interference. To avoid this problem, SoftRate uses a heuristic to identify and separate bit errors caused by interference, adapting the bit rate only in response to the *interference-free BER* of a frame.

If the interferer's signal starts after the receiver synchronizes with the sender's frame, then the interference will manifest itself as a sudden spike in the BER estimated from SoftPHY hints. (If the interferer's signal is much stronger than the sender's, some PHYs will even resynchronize with the interferer and *abort* the sender's frame, which can also be used as a sign of interference.) Assuming that the PHY does not lose synchronization, a sudden change in BER by orders of magnitude within a small number of bits can be used as a sign of interference because such a change cannot be explained by stochastic channel fading in real-life environments, whose physics are more gradual. For example, Figure 4-2 contrasts the patterns of SoftPHY hints for a frame that was in error due to a collision and a frame that had bit errors resulting from fading in a mobile channel.

We will now describe the heuristic more formally in the context of our SoftPHY-capable OFDM PHY prototype described in Section 3.3. While mapping coded data onto subcarriers in one OFDM symbol, the OFDM transmitter interleaves the data onto non-adjacent (in frequency) OFDM subcarriers. This frequency-domain interleaving mitigates bit errors from frequency-selective fading, which causes adjacent subcarriers to fade simultaneously.

Frame size of S_1	**Frame size of S_2**	f_1	f_2
1400 bytes	1400 bytes	12%	12%
100 bytes	1400 bytes	14%	1%

Table 4.1: Fraction of frames f_1 and f_2 at the two hidden terminal senders S_1 and S_2 for which both the preamble and postamble are lost due to interference and are undetected at the receiver. The two senders are continuously transmitting UDP packets and perform exponential backoff on a collision.

A collision, however, still causes interference on all subcarriers. We therefore detect collisions in such a PHY as sudden jumps in BER between adjoining OFDM symbols. Suppose we receive a frame of S OFDM symbols, each symbol containing N_{bps} bits, for a total of $N = N_{bps} \cdot S$ bits, with corresponding SoftPHY hints s_k, $k = 1 \ldots N$. First, we compute p_k from each s_k using Equation 3.21. Then, we average p_k, N_{bps} bits at a time, to obtain S average BERs $\overline{p}_j$, one for each OFDM symbol j:

$$\overline{p}_j = \frac{1}{N_{\text{bps}}} \sum_{i=1}^{N_{\text{bps}}} p_{i+(j-1)\cdot N_{\text{bps}}}. \tag{4.1}$$

Finally, our collision detection algorithm is a simple threshold on the difference $d_j = \left|\overline{p}_j - \overline{p}_{j-1}\right|$. A SoftRate receiver uses this heuristic to test every received frame for the presence of interference, and computes the BER of the frame over the interference-free portions alone.

If the PHY uses time interleaving of bits in a frame, then the bit errors that occur due to interference will be dispersed all over the frame. In such cases, interference detection must be performed before deinterleaving to capture the temporal patterns of bit confidences. If the deinterleaving occurs before decoding (i.e., before SoftPHY hints are generated), then the interference detection algorithm can work on the inputs to the decoder as well. We note that if the PHY uses some form of interference cancellation [22, 19], then the interference detection strategy remains the same, though the fraction of time interference-related losses occur may be lower.

Note that this heuristic only detects interference that starts after the receiver has synchronized with the signal of interest from a sender. On the other hand, if the sender's signal starts after the receiver synchronizes with the interferer's frame, then the receiver will neither detect the sender's frame nor send the BER feedback to the sender. We call such losses *silent losses*. The sender, on not receiving any feedback from the receiver, does not know if the loss is due to weak signal at the receiver that prevented the sender's frame from even being detected or due to a collision. To avoid this confusion, one can add a synchronization "postamble" to the end of every frame (an idea used in previous work, e.g., [27]) which enables the receiver to detect with high probability the portion of the sender's frame that lasts after the interference has ended. When postambles are used, the SoftRate sender can assume that consecutive silent losses indicate a weak signal at the receiver and reduce the transmit bit rate.

The perceptive reader may note that in the cases when the frame durations of the interferer and sender are different (as is likely in multi-rate settings), the sender's frame may

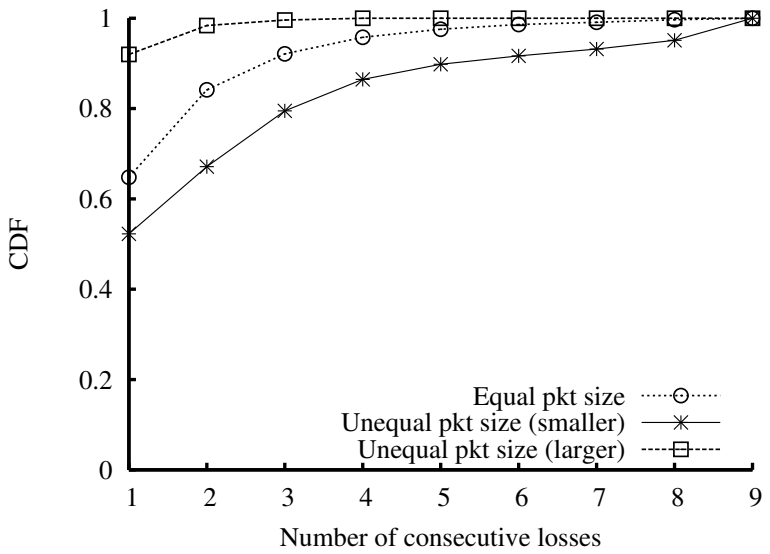


Figure 4-3: Complementary CDF of run length of consecutive frames whose preamble and postamble are undetected at the corresponding receiver in an experiment with a pair of hidden terminal senders.

fully overlap with the interferer's, resulting in a loss of both the preamble and the postamble. However, we observe that such a situation is unlikely to repeat on a retry of both the frames, because channel access protocols typically implement a backoff mechanism on a frame loss, which changes the relative alignment between the frames on the retry.

To measure the frequency of silent losses due to interference, we use the ns-3 network simulator to simulate collisions between two nodes that cannot carrier sense each other. We modify the ns-3 802.11 protocol to append and detect postambles at the end of frames. In our simulation, the two senders S_1 and S_2 transmit UDP packets as fast as possible, picking a random transmit bit rate on each packet. The physical layer parameters of the simulation are set such that only collisions result in frame losses, i.e., there are no noise losses. For each sender S_i, we measure the fraction of frames sent, f_i, for which neither the preamble nor postamble is interference-free (and hence not decodable) at the corresponding receiver. Table 4.1 shows the fractions f_i for simulations with different frame sizes of the two senders; we find that this fraction is under 15% always. For this small fraction of frames that did lose both the preamble and postamble, Figure 4-3 shows the complementary CDF of the run length of consecutive losses at the receivers. We infer from the figure that long runs of losses (say, of length 3 or more) are very uncommon due to interference alone. Therefore, a SoftRate sender assumes that three consecutive silent losses indicate a weak signal at the receiver and lowers the transmit bit rate.

4.2.2 Rate Selection Algorithm

On receiving the feedback of the interference-free channel BER from the receiver in the ACK frame, the SoftRate sender runs a rate selection algorithm to pick the best transmit bit rate for the next packet. A strawman description of the algorithm is as follows. Using

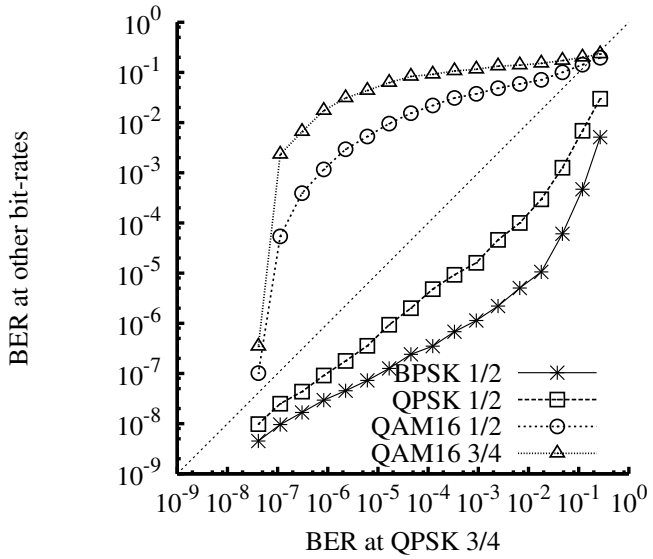


Figure 4-4: BER at the QPSK 3/4 rate vs. BER at other bit rates from Table 3.1, using data from the static and walking traces in Table 3.3.

the BER estimate at one bit rate, the sender guesses the BER at all the other available bit rates. Given the channel BER at a bit rate and a knowledge of what the error recovery scheme does, the sender can compute the link-layer throughput at each of the bit rates, and then proceed to pick the rate with the highest throughput. Of course, we use some approximations to the above description to make the algorithm practical. Below we describe the actual algorithm in its full detail.

The following are the three key mechanisms in the SoftRate rate selection algorithm.

- **BER prediction heuristic.** The channel BER from SoftPHY hints characterizes the channel only at the current bit rate, but rate adaptation requires knowing the channel BER at other bit rates as well. Therefore, a SoftRate sender first uses a heuristic to predict channel BER at a few other bit rates using the BER estimate at one bit rate.

- **Optimal threshold computation.** Using the above BER prediction heuristic, the sender pre-computes *optimal thresholds* α_i and β_i for each rate R_i such that, when the BER at rate R_i is in the range (α_i, β_i), then R_i is the optimal transmit bit rate. That is, rate R_i provides the maximum link-layer throughput based on the predicted BERs at the various rates. Of course, the computation of the thresholds depends on the error recovery mechanism employed by the link layer because the mapping between BER and throughput is determined by the error recovery scheme. Pre-computing the optimal thresholds avoids repeating the throughput calculations at each bit rate for every packet.

- **Bit rate adjustment.** Given the interference-free BER estimate from the receiver and optimal thresholds at each bit rate, the SoftRate sender adjusts its bit rate in the direction of the optimal rate.

Note that SoftRate works when conditions experienced on the upcoming transmission are similar to those on the previous transmission. A wide variety of situations satisfy this criterion, as we discuss later in Section 4.2.3.

We will now describe the above three mechanisms in more detail.

BER prediction heuristic. If one knew the detailed relationship between the BER and SNR for each bit rate, then the problem of predicting BER at multiple bit rates using the BER at the current rate would be an easy one. One could simply look up the SNR corresponding to the BER for the current bit rate, and then consult the various SNR-BER curves to determine the BER at each of the other rates. Unfortunately, because the SNR-BER curves depend on the characteristics of the propagation channel, this method is unlikely to work robustly across different environments.

Instead of relying on SNR-BER relationships, SoftRate uses the two observations below to predict BER.

1. At any SNR, the BER is a monotonically increasing function of the bit rate.

2. There exists a number Δ such that, within the BER range that a bit rate is usable (i.e., BER below 10^{-2}), its BER at a given SNR is at least a factor of Δ higher than that of the next lower bit rate.

We will show that the approximate BER prediction using the above two observations is sufficient for SoftRate's operation, obviating the difficult problem of estimating the SNR-BER relationships.

Let us first see why the above two observations are true. The first observation is well-known and is used by many other rate adaptation protocols. In cases when this assumption does not hold for a communication system, one can come up with orderings of bit rates by the channel BER under various channel quality regimes, and pick the appropriate ordering based on the feedback BER estimate.

The second observation is general enough to hold in practice, independent of radio and environment characteristics. To see why, note that system designers avoid redundancy in bit rates and offer a set of rates that have a considerable difference in error performance at a given SNR. Even if a few adjacent bit rates are similar enough that the channel BER at those rates is almost always the same, the rate adaptation algorithm can always pick a subset of rates with different BERs and use those rates alone for rate adaptation.

Now, how does one determine the value of Δ for a communication system? One can compute Δ theoretically or using empirical data. For example, one can plot the theoretical SNR-BER curves for the various bit rates under a variety of channel conditions (like Figure 3-1) and measure the minimum BER separation between adjacent bit rates across all the graphs. Because the computation of the minimum is done across all wireless channels, the value of Δ will not be sensitive to the propagation environment. Alternately, one can also use empirical data. For example, Figure 4-4 shows a plot of the BER computed from SoftPHY hints at the QPSK 3/4 bit rate, plotted against the BER at two of the higher and two of the lower bit rates. The data for this graph is obtained using our 802.11a/g prototype (Section 3.3) from the static and walking experimental traces described in Table 3.3. One

can see from the graph that when the current best rate is QPSK 3/4 (i.e., the BER at this rate is not as high as, say, 10^{-2}), the BER at the next higher rate is at least a factor of 10 higher, and the BER at the next lower rate is at least a factor of 10 lower. Therefore, a value of $\Delta = 10$ works for the 802.11a/g-like PHY we implemented.

Once the value of Δ is known for a communication system, the BER prediction heuristic is quite simple. If the BER at the current best rate R_i is b_i, then we presume that the BER at the next higher rate R_{i+1} is at least $\Delta \cdot b_i$ and the BER at the next lower rate R_{i+1} is at most $\frac{b_i}{\Delta}$.

Computing optimal thresholds. SoftRate uses the BER prediction method described above to compute optimal thresholds for each bit rate. For each available rate R_i, SoftRate computes α_i and β_i such that, if the BER at R_i is in the range (α_i, β_i) then R_i is the optimal bit rate. Computation of these thresholds depends on the link layer's error recovery mechanism; we will illustrate how to compute thresholds for two such error recovery mechanisms.

Consider the computation of optimal thresholds for the 802.11a/g 18 Mbps bit rate with frame-level ARQ. If the next lower bit rate is 12 Mbps, then until the BER gets to the point where the frame loss rate is 1/3, the sender should remain at 18 Mbps. For a packet size of 10000 bits, that BER would be of the order 10^{-5}. (Note that the BER b and frame loss rate f are related by the equation $f = 1-(1-b)^s$, where s is the frame size in bits and assuming independent bit losses in a frame. The assumption of independent bit losses is valid even though channel fading tends to cause burst errors because wireless systems often use some form of interleaving that makes burst losses appear as independent losses.) Now, if the BER at 18 Mbps is lower than, say, 10^{-7}, then the next higher rate of 24 Mbps is likely to have a low enough BER to see no frame losses and hence have a higher throughput. Therefore, the optimal thresholds for the 18 Mbps rate would be $(10^{-7}, 10^{-5})$.

In contrast, for some smarter ARQ scheme that can recover from a few bit errors easily by retransmitting a small number of parity bits, the throughput at 18 Mbps may be higher than that at 12 Mbps for up to a much higher BER, say, 10^{-3}. The optimal thresholds for such a link layer would be set to $(10^{-5}, 10^{-3})$.

The use of BER as the information signal helps SoftRate integrate cleanly with many different kinds of error recovery protocols, only requiring a recomputing of thresholds to work with a different error recovery scheme. Frame-based protocols lack this modularity because they consider the frame loss rate in making their decisions, tacitly assuming that entire frame retransmissions are used to recover lost frames. As a result, the bit rate adaptation mechanism itself would have to change if the error recovery protocol changed. Architecturally, our proposal decouples rate adaptation from error recovery and separates the two distinct concerns.

Bit rate adjustment. Given the optimal thresholds, SoftRate's rate adjustment algorithm is as follows. Let the current transmit bit rate at a sender be R_i, and let b_i be the most recent interference-free BER estimate at R_i obtained from the receiver. By the design of optimal thresholds, if $b_i < \alpha_i$, then the throughput at the next higher bit rate R_{i+1} will exceed the throughput at rate R_i. Conversely, when $b_i > \beta_i$, the throughput at rate R_{i-1} will exceed

that at rate R_i. Therefore, the sender increases bit rate if $b_i < \alpha_i$, lowers bit rate if $b_i > \beta_i$, and does nothing if $b_i \in [\alpha_i, \beta_i]$.

If b_i is far from the range $[\alpha_i, \beta_i]$, then we can do better by jumping multiple levels to a better bit rate. In the example above, if the BER at 18 Mbps is above 10^{-2}, then one can jump two rates lower to find a bit rate that has a BER under 10^{-5}, as exemplified in Figure 4-4. In general, one can find n levels of rate increase and decrease thresholds α_i^n and β_i^n for every rate i, using which the algorithm jumps n bit rates at a time in the direction of the best bit rate. Our implementation does up to two rate jumps at a time.

4.2.3 Behavior Over Time-Varying Channels

We now discuss why the algorithm described above works well across a wide range of wireless propagation environments. Recall that there are two main sources of variation in the sender's signal at the receiver:

1. Changes in the attenuation or shadowing of the signal, often due to changes in distance between the sender and receiver.

2. Multipath fading, the result of the combining of multiple copies of a signal differing in phase or frequency due to mobility of the transmitter or receiver or objects in the environment.

As explained in Chapter 2, the coherence time of the channel is approximately the duration of time over which multipath fading effects are expected to stay the same. In a slow fading channel, which occurs at walking speeds (or even when the nodes are static, but objects in the environment aren't), coherence times are tens of milliseconds long. In such a channel, the sender's signal fades sharply once every 10-100 milliseconds, typically resulting in a burst of frame losses at higher bit rates, as we saw in Figure 2-1. That is, fading and attenuation happen at a timescale corresponding to multiple frame transmissions at the bit rates and frame sizes in 802.11-like networks. In response to channel changes due to fading and attenuation, SoftRate lowers the bit rate quickly; it also adapts upwards quickly, soon after conditions become better.

Fast fading channels occur at vehicular speeds, where the channel coherence time is between a few tens to a few hundred microseconds, as we saw in Figure 2-2. This duration is shorter than the transmission time of a frame. However, even in this environment, the BER measured by SoftPHY hints accurately reflects the true BER of the channel, as we saw in Section 3.4.3. As a result, SoftRate converges to the best transmit bit rate that maximizes throughput in the presence of the fast fading-induced bit errors over a frame, and adapts this best bit rate in response to changes in attenuation.

Bit rate adaptation is a very hard problem when the coherence time of the channel is in between the two extremes of fast and slow fading, say, equal to two or three frame durations [65]. The way to adapt bit rate in such cases using SoftRate is to increase the packet size to turn it into the fast fading case, or decrease the packet size to turn it into the slow fading case (provided that the packet size is big enough to make this feasible).

4.3 Implementation

Implementing SoftRate requires changes to two layers in the networking stack—the PHY has to change in order to compute and export SoftPHY hints to the link layer, and the actual SoftRate protocol (i.e., the functions of BER estimation, interference detection, sending and receiving the BER feedback, and selecting the best transmit bit rate using the feedback) is implemented in the link layer. We use the implementation of the SoftPHY-enabled OFDM transceiver on software radios (Section 3.3) as the PHY in SoftRate.

However, the high latency (tens to hundreds of microseconds) incurred in both procuring RF samples from the GNURadio front-end and sending link-layer BER feedback makes it impractical to implement and evaluate SoftRate entirely using this platform. We therefore simulate SoftRate and other rate adaptation algorithms in the link layer of the ns-3 network simulator. However, to keep the simulations realistic and to obtain SoftPHY information on receptions, we replace the ns-3 physical layer models with packet traces collected from our live software radio experiments described in Table 3.3.

For each wireless link being simulated, we seed the simulator with a set of traces, one per bit rate, that completely specify the channel characteristics of the link (like, whether a frame sent is correctly received, and what its SNR and SoftPHY hints would be) for each point in time during the simulation. When the PHY in the simulator receives a frame at a certain bit rate, the fate of the frame is determined by looking up the appropriate trace. The bit rate adaptation protocol at the link layer receives and reacts to the feedback from the PHY (frame reception events, SNR estimates, or SoftPHY hints, as the case may be) and sets a suitable bit rate for the next frame. We make no assumptions on the symmetry of links, and use different traces for each of the two uni-directional links between every sender and receiver.

When collecting traces to use in simulations, we ensure that the channel conditions are consistent across the various bit rates at any point of time. For traces collected using the channel simulator, we simulate the same fading process across experiments at different bit rates. We run live experiments in the short range mode (see Section 3.4.1) with small frames sent at each of the bit rates in a round robin manner, running through all the bit rates once in under 5 milliseconds. We find that the BER across the various bit rates is monotonic in 96% of such 5 ms cycles, indicating that the channel is indeed fairly invariant across all the bit rates in a 5 ms snapshot.

All traces are collected with one sender transmitting at a time. In simulations with more than one sender, these traces collected without interference accurately model frame receptions when there are no concurrent transmissions. In case more than two senders transmit simultaneously (e.g., experiments in interference-dominated channels in Section 4.4.5), we assume both colliding frames are lost.

We modify the ns-3 802.11 acknowledgment frame structure to include a 32-bit estimate of the received frame's interference-free bit error rate. We also simulate postamble detection; when this option is enabled, the receiver sends an acknowledgment even if the preamble is not detected but the postamble is interference-free.

The implementation of a SoftPHY-capable PHY in hardware convinces us that the design of SoftRate is practical enough to be implemented in real systems. Recall from the discussion in Section 3.3.2 that the hardware PHY can pass up SoftPHY hints about 8 μs

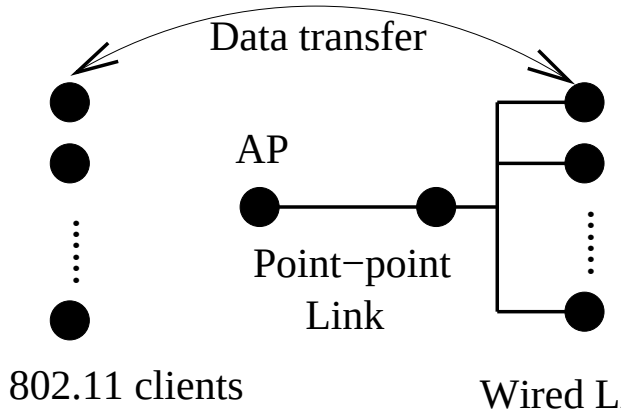


Figure 4-5: Topology used for the ns-3 evaluation of SoftRate.

before the link-layer ACK corresponding to a received frame needs to be transmitted. This slack is more than sufficient for the link layer to embed the BER estimate in the link-layer ACK. We defer a complete real-time hardware implementation of SoftRate to future work.

4.4 Evaluation

This section presents the evaluation of the accuracy of the interference detection heuristic of SoftRate and the trace-driven simulations of SoftRate on ns-3, as described in Section 4.3. In the evaluation of SoftRate, we quantify the performance gains for end-to-end TCP transfers when running SoftRate at the link layer in slow fading mobile channels, simulated fast fading channels, and interference-dominated channels. Our main findings are summarized below.

1. In experiments with slow fading channels that occur at pedestrian mobility, SoftRate achieves 20% more throughput than SNR-based protocols trained for the operating environment, and 2×–4× more throughput than frame-based protocols.

2. In simulated fast fading channels that occur at vehicular mobility, SoftRate achieves from 35% to 4× higher throughput than SNR-based protocols that are not trained for the operating environment.

3. SoftRate's interference detection algorithm identifies collisions with an accuracy of over 80%, because of which the performance of SoftRate stays close to optimal even in the presence of losses due to interference.

4.4.1 Method

We use TCP throughput as the metric to evaluate SoftRate against other rate adaptation protocols because applications like TCP and VOIP are more sensitive to losses, and therefore require responsive and accurate rate adaptation protocols to function well. While previous

work mostly uses UDP throughput as a measure of performance, we believe that gains obtained on UDP transfers without congestion control are hard to realize in most practical applications.

Simulation topology. The topology used in our simulations is shown in Figure 4-5. N clients connect to an access point (AP) that supports the 802.11a/g bit rates from 6 Mbps to 36 Mbps. The AP is connected to a LAN gateway node by a point-to-point link of bandwidth 50 Mbps and one-way delay of 10 ms. In each experiment, N TCP flows are set up to transfer 1400 byte data frames in either direction between the 802.11 clients and the corresponding wired LAN nodes. Each node's link-layer queue length slightly exceeds the bandwidth-delay product of the bottleneck wireless link.

Protocols evaluated. We compare the performance of SoftRate against the following rate adaptation protocols.

1. Two SNR-based protocols that pick bit rates based on the SNR measured at the start of the transmission using the preamble: (i) **RBAR**, a protocol that uses SNR feedback sent via the link-layer ACK to pick the transmit bit rate, without the RTS/CTS overhead as proposed in the original paper [23], and (ii) **CHARM**, a protocol that uses the average SNR over multiple frames [29]. Our simulation does not rely on the channel reciprocity assumptions used in the original protocol because we can afford to change the 802.11 link-layer ACK in the simulator to piggyback SNR information, while CHARM aims to work with existing 802.11 cards.

2. **SNR-Pilot**, an SNR-based protocol that uses the average SNR over the entire frame computed using pilots, as described in Section 3.1.1.

3. Two frame-based schemes: (i) **RRAA**, and (ii) **SampleRate**. The various parameters in these protocols are set as described in the corresponding references, except for the interval over which transmission time averages are computed in SampleRate, for which a value of one second gave a better performance than the ten second value suggested in [11].

4. An **omniscient** protocol that always picks the highest rate guaranteed to succeed, which the simulator with a priori knowledge of channel characteristics computes from the traces.

The SNR-based protocols in each simulation were calibrated using the SNR values and the corresponding channel error rates in the PHY traces of the simulation, as is the common practice. Calibration of the protocols involves computing a mapping from the measured SNR to the throughput at each SNR, and eventually to the best transmit bit rate that maximizes throughput. The calibration process can possibly be improved by using traces with "ideal" SNR estimates (for example, SNR measured using a large number of pilot symbols), in order to reduce the impact of noisy SNR estimates on the calibration process; we defer such an evaluation to future work.

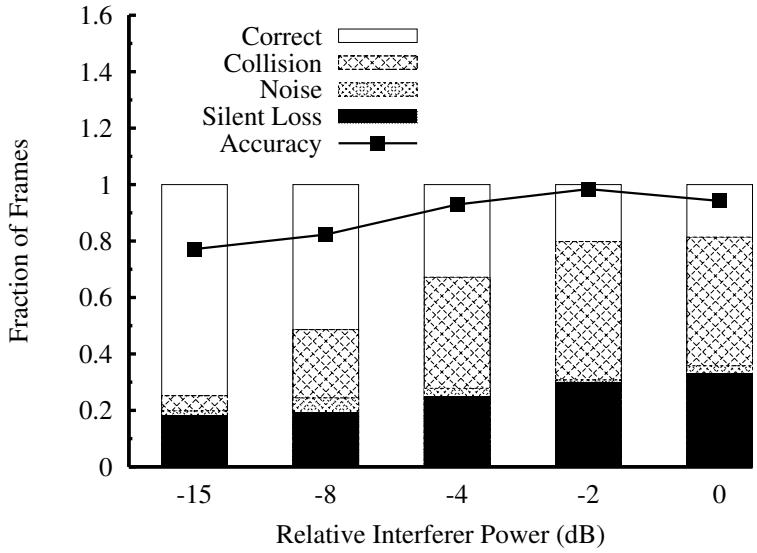


Figure 4-6: Interference detection accuracy as a function of varying interferer power.

4.4.2 Interference Detection Accuracy

We first evaluate the accuracy of our SoftPHY-based interference detection algorithm. We use the SoftPHY hints from various packet traces and run the interference detection algorithm offline. We then compare the result of the algorithm from the ground truth about the presence or absence of collisions in the trace.

False positives. To measure the false positive rate (i.e., the rate at which the fading effects of the wireless channel are falsely identified as collisions), we collect the static and walking traces from Table 3.3 in a quiet frequency band without any other 802.11a/g transmissions. Out of the resulting frames lost, our collision detection algorithm identified less than 1% of them as collisions. This shows that our algorithm rarely mistakes fading losses for interference.

Interference detection accuracy. We use the traces from the static interference experiment described in Table 3.3 to measure the accuracy of our interference detection algorithm. We pick one sender transmit power in the trace corresponding to the sender-receiver link delivering 100% of its frames correctly in the absence of interference. In the presence of interference, one of three things can happen to a frame. First, the frame can be silently lost if the interferer transmits before the sender, either because the receiver has locked on to the interferer's frame, or because the sender's preamble is corrupted by the interferer's signal. Second, the frame can be received, but with errors. Finally, the frame can be correctly received. In the case of frames received with bit errors, we run our interference detection algorithm on the SoftPHY hint traces of the frame to see what fraction of these losses our algorithm identifies as collisions.

We slice the interference detection accuracy results by the different transmit power levels of the interferer and the transmit bit rate of the sender. Figure 4-6 shows the fraction

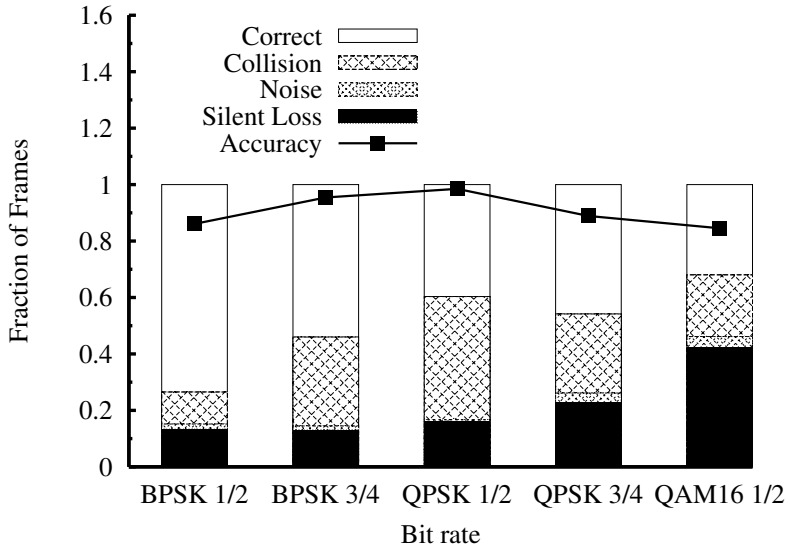


Figure 4-7: Interference detection accuracy as a function of transmit bit rate.

of frames that fall into each of the cases described above versus the relative interferer strength (in dB). Also shown on the graph is the interference detection accuracy of our algorithm, which is computed as the fraction of frames received with bit errors (i.e., the frames corresponding to "collision" and "noise" in the figure) that our algorithm correctly identifies as collisions. Figure 4-7 shows the same data, but broken down by the sender's bit rate. We omit here results for QAM16 3/4 rate, because our current implementation of that bit rate is untuned. We find that our algorithm can always identify more than 80% of frames received in error as collisions. Because the colliding packets are of the same size in this experiment, we will be able to detect most of the silent losses as collisions as well by adding postambles.

4.4.3 Slow Fading Mobile Channels

In this section, we evaluate how well SoftRate can adapt to channel variations that occur at walking speeds in a slow fading channel. We simulate $N = 1, \ldots 5$ TCP flows from the 802.11 clients to the corresponding wired LAN nodes. We use the ten walking traces (Table 3.3) to model the ten uni-directional links. We assume perfect carrier sense among all senders.

Figure 4-8 shows the aggregate TCP throughput obtained by the various rate adaptation protocols as a function of the number of flows. We find that SoftRate outperforms all other protocols, and comes closest to the omniscient protocol. SoftRate gets up to 20% higher throughput than both the SNR-based protocols (RBAR and CHARM) trained over the traces because the BER prediction from SNR of a single packet has a higher variance than that using SoftPHY hints. We also found that using averaged SNR information in CHARM leads to lower responsiveness to short-term SNR variations and hence a slightly worse performance than using just the instantaneous SNR value. The performance of the SNR-Pilot protocol was slightly worse than that of RBAR in the slow fading channel, and

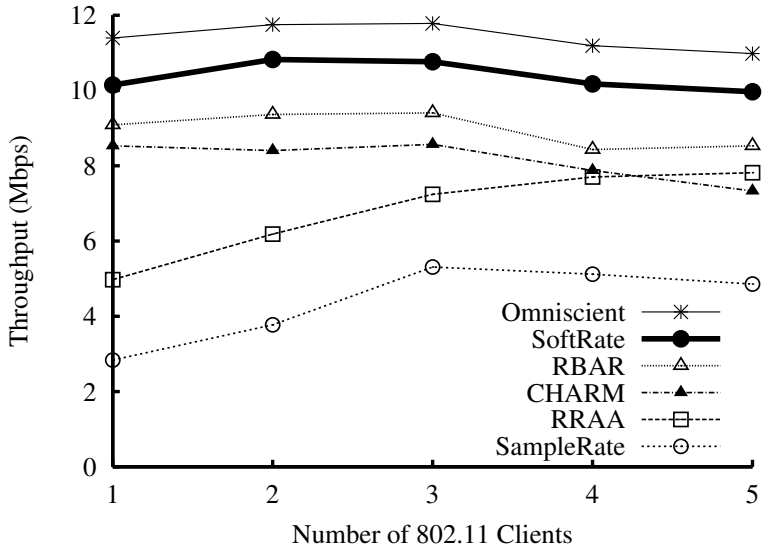


Figure 4-8: Aggregate TCP throughput vs. the number of wireless clients in a channel with slow fading mobility.

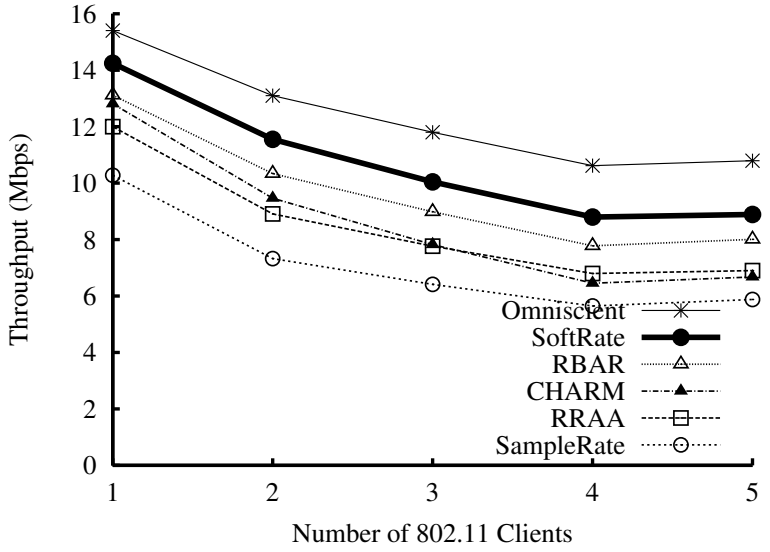


Figure 4-9: Aggregate UDP throughput vs. the number of wireless clients in a channel with slow fading mobility.

is not shown in the figure.

In comparison with frame-based protocols, SoftRate achieves up to $2\times$ higher throughput than RRAA and almost $4\times$ higher throughput than SampleRate because frame-based protocols cannot adapt fast enough to channel fades that are caused due to mobility, with the result that TCP ends up loosing multiple packets in a window and reduces its offered load. We find that the loss rate experienced by TCP is an order of magnitude higher with frame-based protocols than it is with SoftRate. We repeat the simulations with clients

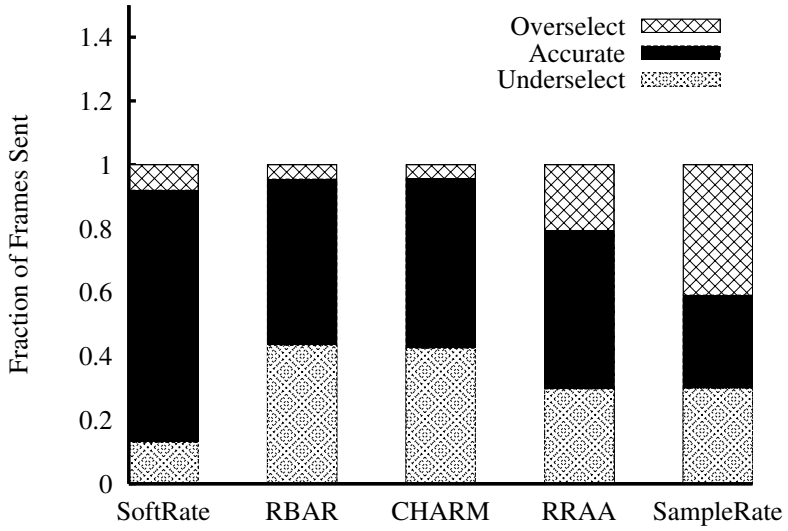


Figure 4-10: Rate selection accuracy with one TCP flow in a mobile slow fading channel.

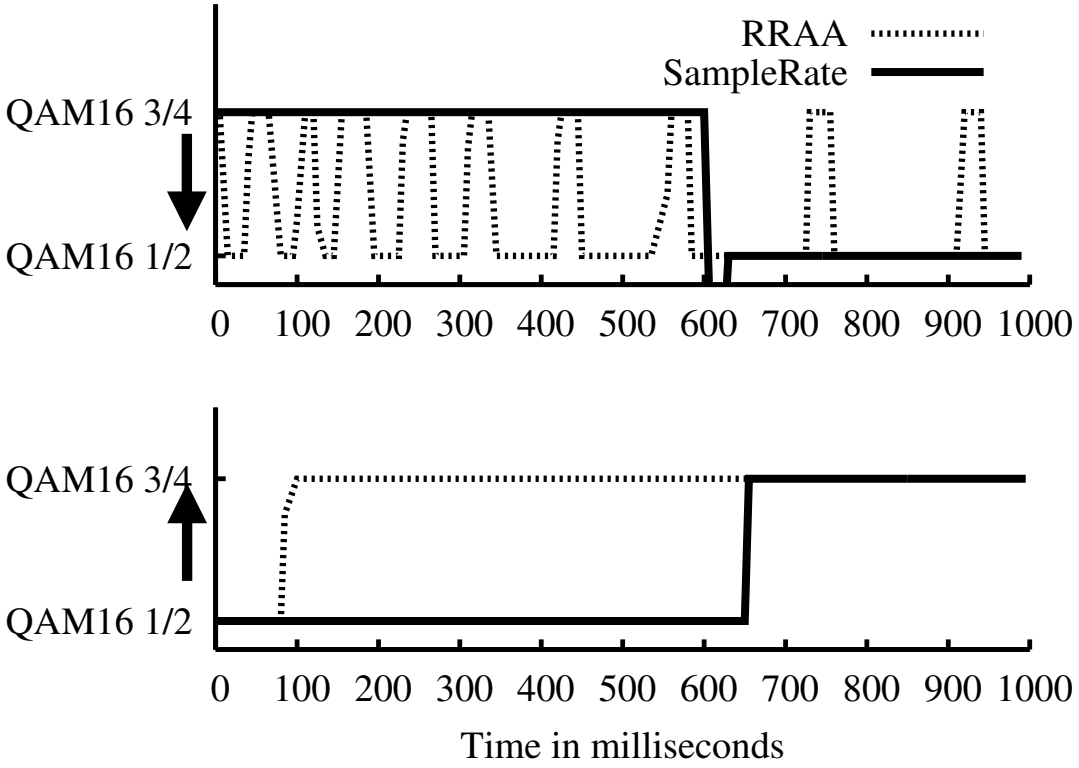


Figure 4-11: Bit rates chosen by RRAA and SampleRate where the optimal bit rate changes at $t = 0$: from a higher rate to a lower rate (*top*), and from lower to higher (*bottom*).

receiving TCP traffic as opposed to uploading TCP traffic; results are similar to those described above. The results for the simulations where the clients send UDP traffic via the access point are shown in Figure 4-9; we notice that the performance trends of the various protocols are similar as before, though the gains due to SoftRate are less pronounced because UDP traffic is less responsive to packet losses.

We now measure the rate selection accuracy of the various protocols. For the simulation with one TCP flow, Figure 4-10 shows how the bit rates picked by the various protocols on every transmitted frame compared against the highest bit rate that would have gotten the frame through at that time. We find that SoftRate chooses the correct bit rate over 80% of the time and has at least 30% higher accuracy than the other protocols.

To better understand the performance of frame-based protocols, we simulate RRAA and SampleRate using a synthetic trace, where the channel alternates between a "good" state (best transmit bit rate is QAM16 3/4) and a "bad" state (best transmit bit rate is QAM16 1/2) every 1 second. Frame trace data for the good and bad states are taken from appropriate snapshots in the walking trace described in Table 3.3. Figure 4-11 shows the bit rates picked by RRAA and SampleRate as a function of time, where the best transmit bit rate moves from the higher rate to the lower rate in the top panel, and back to higher rate in the bottom panel. The convergence times of RRAA and SampleRate are 15 ms and 600 ms respectively in the first case, and 85 ms and 650 ms in the second. These convergence times explain why the frame-based protocols frequently overselect and underselect compared to the optimal in Figure 4-10. One other interesting point to note is the instability of RRAA's rate choice (see the top panel of Figure 4-11), highlighting another short-coming of frame-based protocols. When the frame loss rate at a bit rate is zero, frame-based protocols have no way of knowing if the frames are barely making it through (i.e., the next rate will not work) or if they are getting through very comfortably (i.e., next rate may work). SoftRate knows what the BER at the current rate is and hence can predict whether the next rate will work or not, obviating the need to unnecessarily probe higher rates.

In summary, we note that failing to adapt the transmit bit rate quickly to channel fades that occur with mobility can lead to burst losses that reduce TCP throughput. As a result, a responsive bit rate adaptation protocol like SoftRate offers huge gains for TCP in mobile channels, compared to less responsive frame-based protocols.

4.4.4 Simulated Fast Fading Channels

In this section, we evaluate the performance of SoftRate in fast fading channels that occur at vehicular mobility speeds. We simulate one 802.11 client transferring TCP data to a wired LAN node via the AP. We use the channel simulation traces from Table 3.3 to model the links.

We present the throughput of the various protocols normalized by the throughput of the omniscient protocol because the best transmit bit rate (and hence the absolute throughput achieved) decreases with channel coherence time. Figure 4-12 shows the normalized throughput of the TCP flow with various rate adaptation protocols as a function of varying channel coherence time. Let us first compare SoftRate with the SNR-based protocol RBAR. In this simulation, the SNR-BER relationships used by this protocol are the same as those obtained from traces collected at walking speed in Section 4.4.3. Recall that the

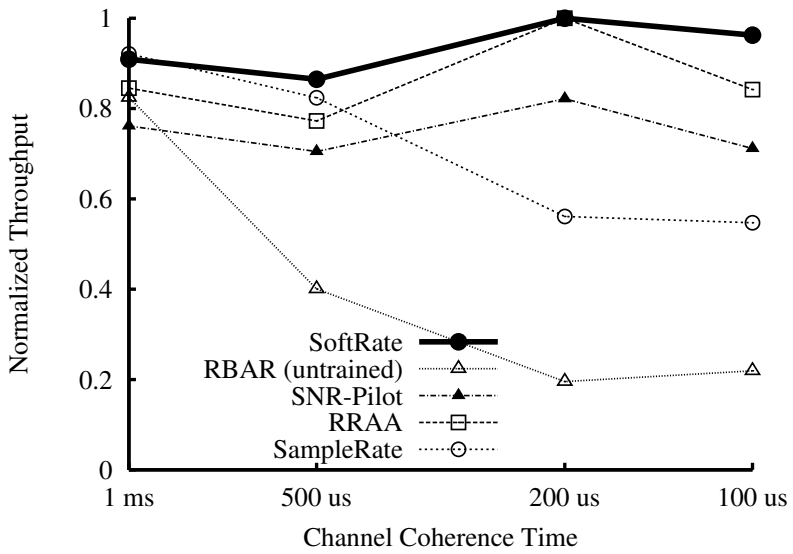


Figure 4-12: Normalized TCP throughput as a function of the channel coherence time in a simulated fast fading channel.

SNR estimate in RBAR is measured at the start of the frame. Therefore, as channel coherence time reduces, the channel BER at any given bit rate increases for the same measured SNR at the start of the frame. As a result, RBAR underestimates the frame BER at lower coherence times and ends up selecting bit rates that are above optimal. Because SoftPHY hints measure the average BER over the entire frame, SoftRate correctly picks the bit rate that codes for the average channel BER in fast fading channels and its performance stays the same across various coherence times even without retraining. We see from the figure that SoftRate achieves a performance gain of about $4\times$ over untrained RBAR at a channel coherence time of 100 μs. Gains over CHARM were similar, as we did not use CHARM's retraining mechanism that adjusts SNR thresholds every few seconds, in order to isolate the impact of training on the performance of SNR-based protocols. Also, such mechanisms are meant to handle calibration issues across different hardware and are ineffective if the coherence time of the channel changes on a short timescale, for example, when a train passes by a stationary user.

On the other hand, the performance of the SNR-Pilot protocol does not deteriorate as the channel coherence time decreases because the SNR-BER curves generated using the average SNR from pilots are not as sensitive to the channel coherence time (Section 3.4.3). However, BER prediction using SNR estimates computed from pilots has a high variance (Section 3.4.2), because of which SoftRate outperforms the SNR-Pilot protocol by 20%–35% at various channel coherence times. SoftRate also outperforms the frame-based protocols because it reacts faster to channel changes.

4.4.5 Interference-Dominated Channels

In this section, we evaluate the impact of interference losses on the performance of SoftRate. The simulation consists of five 802.11 clients uploading TCP data via the AP to the

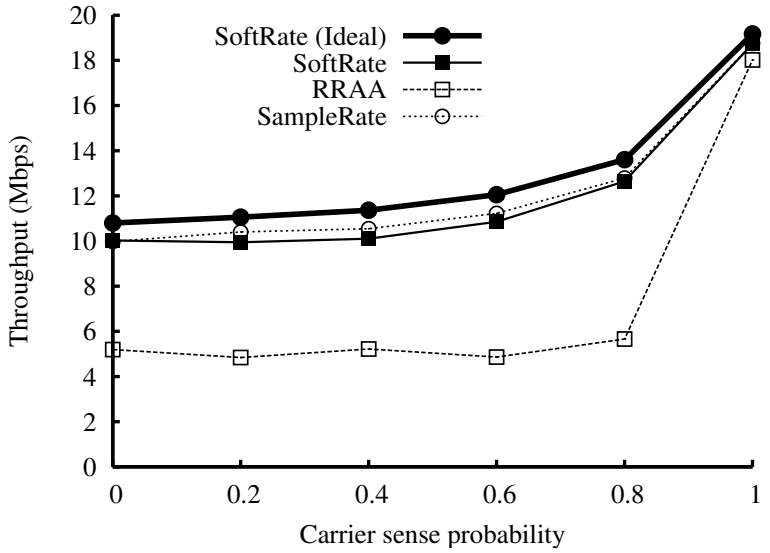


Figure 4-13: Aggregate TCP throughput as a function of carrier sense probability between the senders in a simulation with five wireless clients.

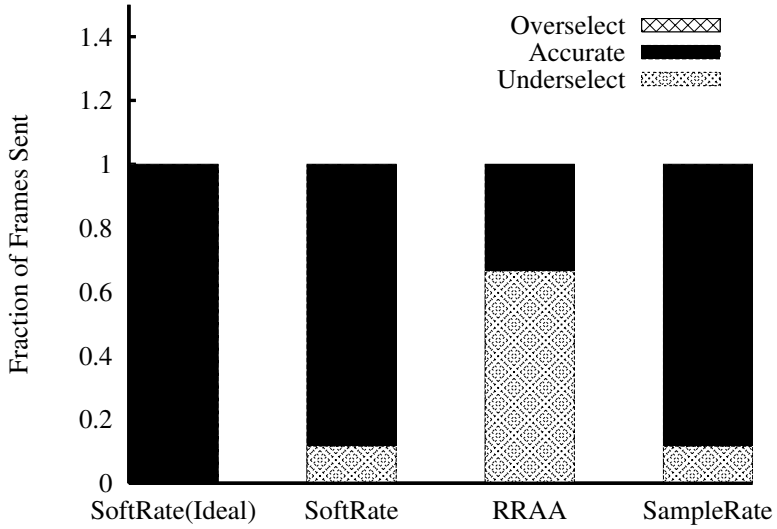


Figure 4-14: Rate selection accuracy with the carrier sense probability between the senders set to 0.8.

wired LAN nodes. We use the static short range traces described in Table 3.3 to model each of the uni-directional links; using a static channel helps us isolate the benefits due to interference detection from those due to better adaptation in mobile channels. We simulate imperfect carrier sense between the various senders in the simulation to generate collisions. We vary the carrier sense probability between the senders from 0 (i.e., all senders are perfect hidden terminals) to 1 (i.e., perfect carrier sense and hence no interference losses). We simulate two versions of SoftRate—a present version where interference detection suc-

ceeds 80% of the time and there is no postamble detection, and a yet-to-be-implemented "ideal" version with postambles and perfect interference detection. When the SoftRate receiver identifies a frame loss as interference, the feedback BER from the receiver is simply the interference-free BER measured in the trace. Otherwise, the feedback is a very high BER indicating a noise loss.

Figure 4-13 shows the performance of the various protocols as a function of carrier sense probability. RRAA, because it reacts to short-term frame loss rate, reduces its bit rate in response to interference and sees a much lower throughput than the other protocols. We found RRAA's Adaptive RTS/CTS scheme to be ineffective in preventing collisions, because interference was unpredictable and resulted in RTS/CTS being constantly turned on and off without any real benefits. SampleRate, on the other hand, is resilient to interference losses because it computes the average transmission time at each bit rate over slower timescales; interference affects the transmission time at all bit rates uniformly at such timescales. The performance of the omniscient protocol is very similar to that of the ideal SoftRate and is not shown. Figure 4-14 shows the rates picked by the various protocols on every transmitted frame, compared against the optimal bit rate choice. As expected, RRAA frequently underselects. The performance of the SNR-based protocols is not considered in these graphs because in the case of RBAR and CHARM, the SNR was estimated using the preamble before interference started and therefore was not sensitive to interference.

In summary, protocols that react to short-term channel variations entail the danger of lowering bit rate on interference losses. SoftRate's interference detection mechanism avoids this penalty.

4.5 Chapter Summary

This chapter presented SoftRate, a cross-layer wireless bit rate adaptation protocol that achieves throughput gains of up to $2\times$ over frame-based protocols such as SampleRate and RRAA, 20% over SNR-based protocols trained on the operating environment, and from 35% to $4\times$ over untrained SNR-based protocols. The key idea in SoftRate is to use the channel BER computed from SoftPHY hints as feedback to pick transmit bit rates. SoftRate reacts to channel changes faster than frame-based rate adaptation protocols because it can estimate the channel BER using just one frame reception. SoftRate does not require any environment-specific calibration, because the BER computed from SoftPHY hints is accurate irrespective of the wireless channel characteristics. The performance of SoftRate does not degrade in interference-dominated channels because patterns of SoftPHY hints can be used to identify and eliminate the effects of strong interference on channel BER estimates. SoftRate highlights the benefits of using fine-grained PHY information to better estimate the wireless channel for rate adaptation, and has inspired recent research on cross-layer rate adaptation protocols [57].

Chapter 5

Harnessing Exposed Terminals Using Conflict Maps

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. However, current contention-based channel access protocols like CSMA 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. This is because the MAC layer relies on very little input from the PHY layer to make channel access decisions—information on whether there is some other ongoing transmission near the sender—and conservatively decides to defer without knowing if the concurrent transmission can succeed or not.

In this chapter, we present a new approach to MAC protocol design using information available via the new streaming SoftPHY interface. There are two components to our solution. First, the *streaming PHY interface*, where the PHY streams received bits to the MAC as soon as they are decoded, enables the MAC to identify the senders and receivers of ongoing transmissions and make smarter channel access decisions based on who else is transmitting. Second, senders use a *conflict map*, a data structure that indicates which transmissions can proceed in parallel and which cannot, to make channel access decisions.

We describe two protocols based on the idea of conflict maps—SoftCMAP and CMAP. These protocols nearly double the throughput of exposed terminals and increase the number of successful concurrent transmissions in the network. Both protocols are completely distributed and do not rely on a centralized scheduler or controller for their operation. We begin the chapter with experiments that evaluate the potential improvements possible by exploiting concurrency in data networks (Section 5.1). We then describe a high-level overview of the conflict map idea (Section 5.2) and then describe each protocol in more detail (Sections 5.3 and 5.4). We evaluate the conflict map protocols in Section 5.5. We also compare our channel access schemes with related work in Section 5.6.

5.1 Motivation

We describe two experiments to understand the opportunities for concurrent transmissions in data networks, and the gains that can be had by cleverly exploiting such opportunities.

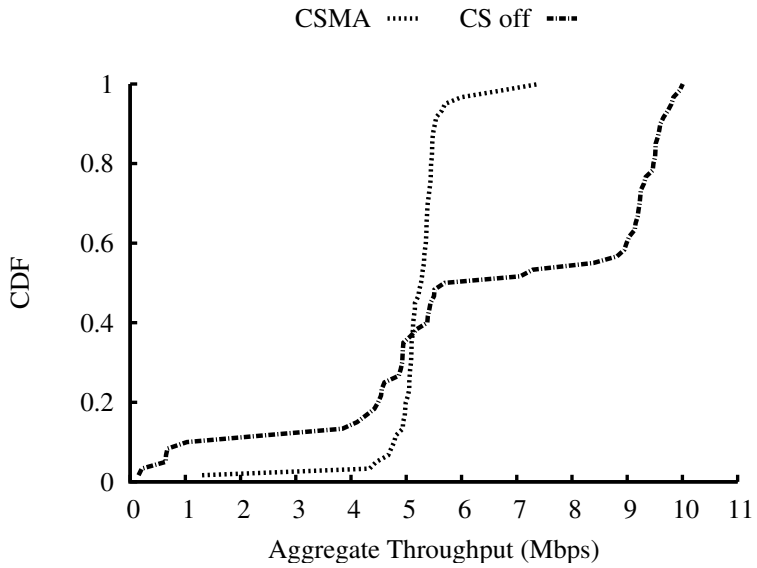


Figure 5-1: Aggregate throughput of two receivers in an experiment with two 802.11a senders transmitting at 6 Mbps, repeated with carrier sense turned on and carrier sense turned off. The CDF is plotted over 50 experiments with different sets of nodes.

The first experiment is performed on an indoor 802.11a testbed, and tests the prevalence of exposed terminals in a real network setting. The second experiment considers sets of exposed terminals and measures the maximum throughput gain possible by enabling concurrent transmission between the two senders.

Prevalence of exposed terminals. We perform the first experiment on a 50-node indoor 802.11a testbed described in Section 5.5.1. We consider 50 random sets of four nodes each from the testbed, forming two sender-receiver pairs. The sender-receiver pairs are picked as described in Section 5.5.2. The senders are within hearing range of each other, and the senders have a good transmission link that can deliver over 90% of the transmitted packets to their respective receivers. In each of the 50 experiments, the two senders continuously transmit 1400-byte UDP packets to their receivers at the 6 Mbps rate of 802.11a. The throughput at each receiver is calculated based on the number of packets from its sender that were correctly received. We perform the experiment first with carrier sense turned on between the senders, and then repeat the experiment with carrier sense turned off. When carrier sense is turned off, both senders end up transmitting all the time and their transmissions overlap in time. On the other hand, when CS is on, the senders defer to each other when in carrier sense range of each other.

Figure 5-1 shows the CDF of the aggregate throughput at the two receivers with CS on and off, over 50 experiments with two transmissions each. With CSMA, we find that most experimental runs achieve an aggregate throughput of around 5 Mbps (which is close to the maximum rate achievable by the 6 Mbps bit rate after discounting for MAC-layer overheads). This is because CSMA allows only one of the two senders to transmit at any given time. For a small portion of senders at the right end of the CSMA curve, the

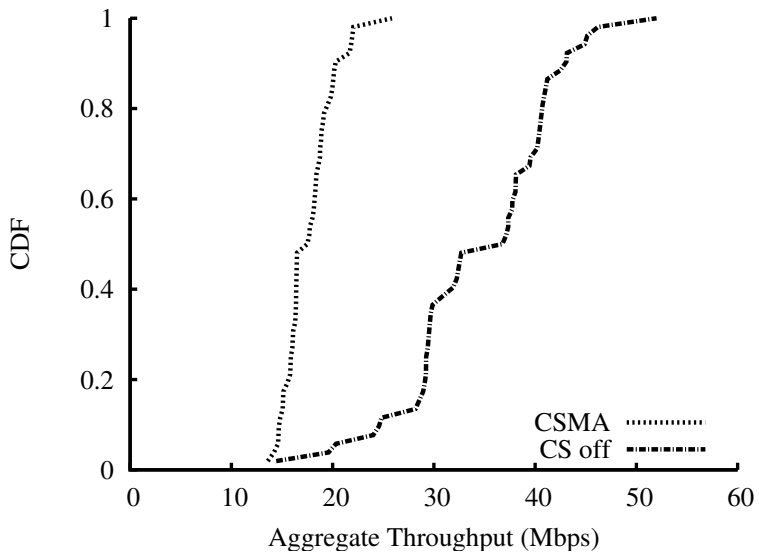


Figure 5-2: Aggregate throughput of two receivers in an experiment with two exposed terminals transmitting at optimal bit rates between 6 and 36 Mbps, repeated with carrier sense turned on and carrier sense turned off. The CDF is plotted over 50 experiments with different sets of nodes.

aggregate throughput is more than 6 Mbps. In such cases, the senders would have been far enough from each other that they are out of each other's carrier sense range, and end up transmitting simultaneously with CSMA. A small fraction of senders at the left end of the CSMA curve see an aggregate throughput below 5 Mbps; these pairs of senders are likely hidden terminals and see huge loss rates due to the carrier sense mechanism not functioning properly. Now consider the curve that corresponds to carrier sense turned off. We find that for about 40% of the sender pairs, the throughput with CS off is much worse than the throughput with CS on, suggesting that these pairs of transmissions conflict with each other and must be performed one at a time. For the remaining pairs however, the throughput with CS off is higher than that with CS on, suggesting that these pairs of senders are exposed terminals. Such pairs of transmissions suffer an unnecessary reduction in throughput when CSMA is used as the channel access scheme.

This experiment shows us that a random pair of transmissions chosen in our testbed has a non-negligible probability of being exposed. Other researchers have also observed that exposed terminals occur frequently in busy access point-based wireless networks [30]. While the exact proportion of exposed senders may vary with the topology of the network, our results show that the proportion may be significant enough to impact overall network throughput if harnessed properly.

Performance gains from exploiting concurrency with rate adaptation. We now investigate the performance gains possible by harnessing exposed terminals, in the presence of heterogeneous bit rates and smart rate adaptation. In particular, we want to answer the following question: do exposed terminals exist only at lower bit rates where there is sufficient

"slack" in the transmitted signal to accommodate interference, or can one gain performance benefits from concurrency even when senders can transmit at higher bit rates?

We perform this experiment with 50 sets two sender-receiver pairs, where the two senders in each set are exposed terminals at the lowest bit rate of 6 Mbps, i.e., the aggregate throughput of the pair of transmissions doubles when carrier sense is turned off. We run this experiment on a testbed of GNURadio-based 802.11a/g-like software radio nodes (described in Section 3.3). Our prototype is capable of running at modulations and coding rates corresponding to the 802.11a/g rates of 6, 9, 12, 18, 24, and 36 Mbps. We could not run this experiment on commodity wireless cards unlike the previous experiment because very few commodity cards allow turning carrier sense off, and the cards that did permit it were not capable of transmitting at bit rates above 18 Mbps robustly.

We run the experiment by making the two senders transmit 1400-byte packets with both carrier sense being turned on and off. Note that because carrier sense cannot be implemented on software radios, we make the two senders transmit one after the other to simulate CSMA. To simulate carrier sense turned off, the senders simply transmit continuously. Because software radios run on a lower bandwidth than commercial devices (our prototype runs on a 500 kHz channel, as opposed to the 20 MHz channel of 802.11), we computed the throughput numbers in this experiment by extrapolating the observed frame loss rate to a 20 Mhz channel. Our software radio prototype does not implement a MAC protocol. Therefore we compute the throughput numbers by accounting for MAC-layer overheads such as inter-frame spacing, MAC header, and link-layer ACKs, using the timing numbers from the 802.11a specification. We repeat each experiment for all possible combinations of bit rates at the two senders, and compute the aggregate throughput of the pair of transmissions as the maximum aggregate throughput achieved across all bit rate combinations. One can assume than an intelligent bit rate adaptation protocol at each sender will converge to this choice.

Figure 5-2 plots the CDF of this maximum aggregate throughput achieved across the 50 pairs of exposed terminals, for experiments with simulated carrier sense turned on and off. The figure shows that some pairs of exposed senders do not see a full $2\times$ improvement on turning off carrier sense when multiple bit rate choices are present. This is because some senders may not be able to transmit at their optimal bit rate when there is added interference from another concurrent transmission. In such cases, the maximum aggregate throughput when carrier sense is disabled will be achieved with one or both of the senders transmitting at a less-than-optimal bit rate. For example, in one experiment with carrier sense disabled, one pair of senders achieved the maximum aggregate throughput when each sender was transmitting at 18 Mbps, whereas each sender was able to send at 24 Mbps with CSMA. However, the aggregate throughput of the pair of transmissions is 36 Mbps when transmitting concurrently, and is still higher than the 24 Mbps with CSMA. Therefore, even though the improvement in throughput on turning off carrier sense is less than $2\times$ (it is in fact 36 Mbps vs. 24 Mbps, i.e., 50%), we find that turning off carrier sense is the wiser choice for this pair of senders. We see from the figure that about 20% of the exposed terminals are "partially exposed" in this manner. The remaining 80% of the exposed sender pairs remain exposed terminals even in the presence of higher bit rates. This experiment shows us that there are huge throughput gains to be had by a smart channel access scheme that cleverly exploits concurrency with some senders selectively, and jointly performs channel access

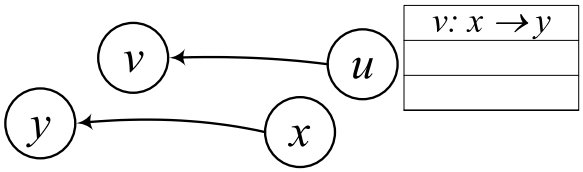


Figure 5-3: An example of a defer table entry at node u indicating that u must not transmit to v when $x \rightarrow y$ is in progress.

and bit rate decisions.

5.2 Design of Conflict Map-Based Protocols

This section provides an abstract description of the two conflict map protocols, SoftCMAP and CMAP. We describe how nodes build the conflict map and use it along with the streaming PHY interface to increase the number of successful concurrent transmissions in the network. The subsequent sections will describe the implementation details specific to each protocol. We use the notation $u \rightarrow v$ to denote a transmission from u to v and $u \rightarrow *$ to denote a transmission from u to any node.

5.2.1 Conflict Map Data Structures

We begin with the definition of a conflict. Note that the conflict map protocols seek to promote concurrency between pairs of senders. Therefore, we define conflicts also between pairs of transmissions. Consider the example of two transmissions $u \rightarrow v$ and $x \rightarrow y$. Nodes u and v consider that $x \rightarrow y$ conflicts with their transmission if they notice that the throughput of $u \rightarrow v$ when transmitting concurrently with $x \rightarrow y$ is lower than the throughput of u when u and x share the medium by taking turns to transmit one after the other. Note that the conflict relation need not be symmetric. For example, x's transmission may interfere with v's reception during the concurrent transmission, while y may be able to receive perfectly during that time.

Note that if nodes use omni-directional antennas and transmit at the same power no matter who the receiver is, then it is sufficient to infer conflicts as a relation between a transmission $u \rightarrow v$ and an interferer x—that the receiver is y is irrelevant, as it does not determine the extent of x's interference at v. Transmit power control or directional antennas are almost never used in 802.11-based data networks today, and we will proceed under this assumption. Our protocol can be easily extended to work correctly if these assumptions do not hold.

We now describe the two main conflict map data structures—the *defer table* and the *rate table*. Each node stores a list of transmissions that conflict with its intended transmissions in its defer table and uses it to make channel access decisions. For example, if the defer table at u has an entry of the form $(v : x \rightarrow y)$, then u must not transmit to v when $x \rightarrow y$ is in progress, as shown in Figure 5-3. The union of all the defer tables at all the nodes in the system completely captures all the conflict information in the entire network. This

union is referred to as the *conflict map*. In other words, the conflict map is a distributed data structure, with each node locally storing a slice of information relevant to its transmission decisions in its defer table. The defer table is populated using feedback from the receivers about the fate of concurrent transmissions, as we will see in Section 5.2.2.

Every sender also maintains a rate table that specifies the transmit bit rate that the node must use when the channel access scheme allows a transmission. The rate table has two types of entries. Node u maintains an entry (v, r_1) for each of its destinations v, specifying the bit rate r_1 to use when transmitting without interference to that destination. It also has an entry (v, x, r_2) for every pair of intended destination v and interfering sender x, specifying the rate r_2 that u must use in transmissions to v when there is a concurrent transmission from x. Note that for any destination v, the rate r_1 is necessarily higher than or equal to rate r_2 because the added interference from x never increases the optimal bit rate.

Note that at a node u, for every pair of destination v and interferer x, there must either exist a conflict entry in the defer table (if the transmissions cannot proceed concurrently) or a transmit bit rate entry in the rate table (if the transmissions can proceed concurrently). The defer table and the rate table can be implemented as hash tables keyed by the identity of the destination and interferer. A typical radio neighborhood will not have more than a few tens of active senders and flows at any time. Therefore the sizes of these data structures are likely to be small and manageable in hardware or software.

5.2.2 Building the Conflict Map

We will now describe how the defer tables are populated. Initially, the defer table at each node is empty, and nodes transmit whenever they have data to send. Nodes then infer conflicts based on the fate of concurrent transmissions as observed at the receivers. This conflict information is subsequently used to populate the defer tables.

To infer a conflict from an interfering sender, the receiver needs two pieces of information: the degradation in channel quality at the receiver due to the concurrent transmission, and the identity of the interfering sender. The channel quality is measured using frame loss rates in the CMAP protocol, and using the BER from SoftPHY hints in the SoftCMAP protocol. Alternately, one can also use other measures of channel quality such as the SNR information to infer conflicts [33]. The mechanism of obtaining the second piece of information, the identity of the interfering sender, is different for the two protocols and will be described in the context of the individual protocols. For now, we assume that when a concurrent transmission happens, either the sender or the receiver can infer the identity of the interfering sender.

An *interferer list* at a node consists of entries of the form $(u \rightarrow v, x)$ indicating that x interferes with transmissions from u at node v. The interferer list is constructed using receiver feedback about channel quality during a concurrent transmission. Conflict map receivers send feedback about channel quality in a feedback frame like the link-layer ACK, similar to how the BER feedback is sent in SoftRate. If the receiver can infer the identity of the interferer as well, then the receiver feedback is the interferer list itself. If not, then the receiver feedback is some channel quality measure that enables the construction of such a list by the sender. Upon constructing interferer lists in this manner, nodes periodically broadcast their interferer lists to all one-hop neighbors, either as standalone packets or by

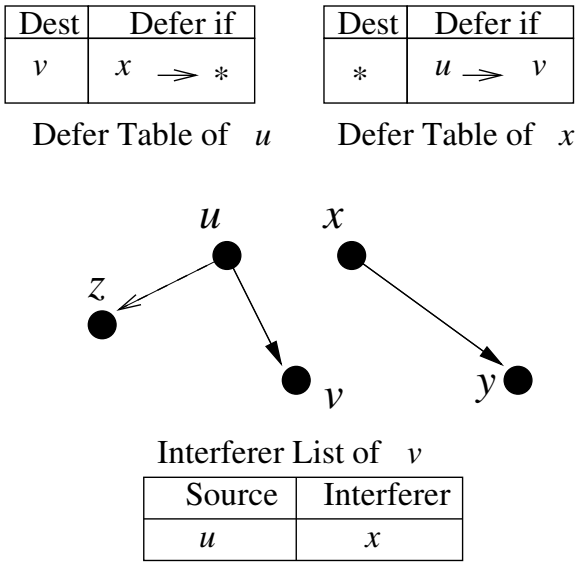


Figure 5-4: Examples illustrating the rules that populate the conflict map data structure.

piggy-backing the interferer lists with routing beacons or other control messages.

We will now see how the feedback in the interferer list is used to populate the defer tables at the various nodes. When a node p receives an interferer list I, it updates its defer table using the following two *local* rules at p:

Update rule 1: $\forall q, r : (p \rightarrow r, q) \in I$ *add* $(r : q \rightarrow *)$ *to the defer table.*

Update rule 2: $\forall q, r : (q \rightarrow r, p) \in I_r$ *add* $(* : q \rightarrow r)$ *to the defer table.*

Update rule 1 states that when q interferes with $p \rightarrow r$, then node p must not send to r whenever q is transmitting the next time to any node. Update rule 2 states that when p interferes with $q \rightarrow r$, then p must not transmit to any other node the next time it sees $q \rightarrow r$ in progress.

To see why these rules make sense, consider the example in Figure 5-4. When u receives feedback from v corresponding to the interferer list entry $(u \rightarrow v, x)$, it adds the entry $(v : x \rightarrow *)$ to its defer table by Rule 1, because u now knows that its transmitting packets to v while x is transmitting to *any* node is likely to degrade the throughput of u. Note that u need not defer while transmitting to all destinations, e.g., it may be able to transmit successfully to some other node z while $x \rightarrow *$ is in progress. Accordingly, Rule 2 does not apply at u. On the other hand, when x receives the above interferer list, it adds an entry $(* : u \rightarrow v)$ to its defer table by Rule 2. Note that x cannot transmit to *any* destination (not just y) while $u \rightarrow v$ is in progress, because its transmission to any node will cause interference at v. On the other hand, x can transmit freely when u is transmitting to a node other than v (say, z) as long as it knows it doesn't cause interference at that node. Accordingly, Rule 1 does not apply at x.

In general, it suffices to broadcast the interferer list to just the one-hop neighbors, because a node does not hear headers from interferers that are more than a hop away and hence does not know about them. However, in networks with asymmetric links (e.g., the receiver

can hear the interferer's headers but the interferer cannot hear the receiver's interferer list updates), it may help to propagate the interferer list over two hops. Nodes periodically time out the entries in their interferer lists and defer tables in order to accommodate changing channel conditions and interference patterns. However, the most suitable deployment scenario for the conflict map protocols would be an infrastructure or mesh network with little or no mobility, where the somewhat expensive process of discovering conflicts can be performed at a coarse granularity of every few minutes.

5.2.3 Checking For Conflicts

Suppose node u wants to send a packet to destination v. We now describe how u checks if its proposed transmission conflicts with another transmission $x \rightarrow y$. First, u checks that v is neither sending nor receiving packets at that moment, i.e., v is neither x nor y. Next, u checks its defer table to see whether one of its entries matches the following patterns that indicate a conflict between the two transmissions:

Defer pattern 1: $(* : x \rightarrow y)$

Defer pattern 2: $(v : x \rightarrow *)$

If no match exists in the defer table, then $u \rightarrow v$ and $x \rightarrow y$ do not conflict and can proceed concurrently. If an entry matches, then the two transmissions conflict and must happen at different times.

5.2.4 PHY Interface

The idea of conflict maps relies on extending the MAC-PHY interface to provide more information about ongoing transmissions so that the MAC can make smarter channel access decisions. In particular, the conflict map protocols make channel access decisions based on who is transmitting to whom, i.e., based on the MAC-layer identifiers of the sender and the receiver that are typically embedded in the MAC header. There are two uses of knowing this information: (a) a node that just finished a failed concurrent transmission can know who else was transmitting and thereby infer conflicts, and (b) a node that is about to start a transmission can know if any other potentially conflicting transmission is ongoing. By using a streaming PHY, where the PHY passes up bits to the MAC as they are decoded, the MAC layer decodes the header and identifies the sender and receiver of the transmission soon after the transmission starts.

Decoding the MAC header before packet reception completes poses a problem of error detection in the header for current 802.11 packets, because the entire packet (including the MAC header and the payload) is protected by one CRC at the end of the packet. Therefore the link layer cannot perform error detection and use the information in the MAC header even if it is streamed up early. To avoid this problem, we protect the node identifiers in the MAC header with a separate CRC in the conflict map protocols, to ensure their verification before using them for channel access decisions.

A streaming PHY alone is not enough to provide all the information a MAC needs in the conflict map protocols. For example, a node that just finished transmitting a packet would

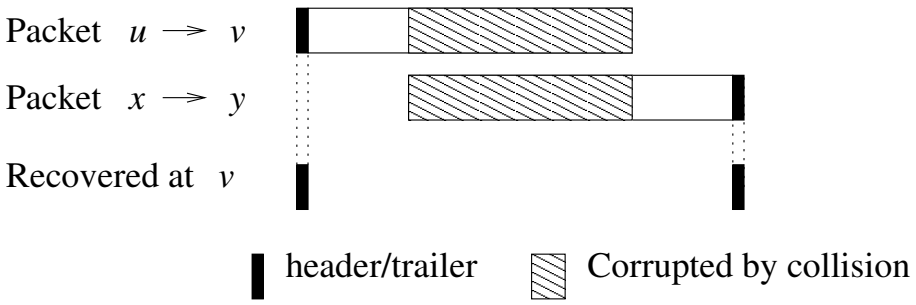


Figure 5-5: In the collision between $u \rightarrow v$ and $x \rightarrow y$, the trailer from x is interference-free and can be decoded, showing that one of the header or trailer from the interfering sender survives a collision with high probability.

have missed the headers of the packets that started during its transmission and cannot know the current transmission on the air. To address this problem, we propose appending a MAC trailer that can be decoded independently from the header at the end of the packet. The trailer contains the sender and receiver MAC identities and other useful information from the MAC header, and is protected by a separate CRC as well. Nodes that were unable to decode the header at the start of the packet will be able to attempt to decode the trailer and learn about the transmission that ended. We will describe the specific implementation and uses of the trailer in the context of the two protocols separately.

5.3 The CMAP Protocol

The CMAP protocol is a MAC protocol based on the idea of conflict maps described in the previous section. We design CMAP with the goal of implementing and deploying a prototype using commodity 802.11 hardware. The commodity 802.11 wireless cards today come with the 802.11 MAC protocol built into the firmware, and any changes to the MAC can be made only by controlling the software driver. Many of the design decisions in CMAP reflect this constraint. Later, we describe the SoftCMAP protocol, which takes a clean-slate approach to MAC protocol design, and is unencumbered by the constraints of today's 802.11 hardware.

5.3.1 Design

Packet format and PHY interface. The current 802.11 hardware does not provide a streaming PHY abstraction: it delivers headers of a frame along with the complete frame only after frame reception has completed. CMAP proposes to approximate the streaming PHY abstraction by sending the header and trailer information in separate frames immediately before and after a "payload" frame respectively. Besides the identity of the sender and receiver, the header and trailer frames contain a sequence number counting the number of frames between that sender and the receiver, and an estimate of the duration of the transmission. The sequence number and duration are used by receivers to infer conflicts.

Building the conflict map using frame loss rates. CMAP uses empirical feedback from receivers about frame loss rates in the presence and absence of interference to infer conflicts. Note that using frame loss rates to measure channel quality has a few disadvantages. First, it takes many measurements to discover conflicts, leading to transient packet losses till the discovery is completed. Second, it takes many *more* measurements to measure the frame loss rates at different combinations of bit rates of the two senders, in order to populate the rate table entries corresponding to the optimal bit rate for concurrent transmissions. Therefore, we assume that all nodes use the same bit rate in CMAP, making the rate table data structure defunct. When all nodes use the same bit rate and perform frame-level error recovery, the throughput at a node is directly proportional to the frame delivery rate.

As described in Section 5.2.2, identifying a conflict requires knowing two pieces of information: the identity of the interfering sender and the degradation in channel quality caused by the concurrent transmission. We now describe how one obtains this information in CMAP. During the concurrent transmission between $u \rightarrow v$ and $x \rightarrow y$, either the header or the trailer from x will be recovered at node v with a high probability. To see why, if the transmission $x \rightarrow y$ starts before $u \rightarrow v$, then v would have decoded the header from x with high probability. If $u \rightarrow v$ starts first, then v would have decoded the interference-free trailer from x, as shown in Figure 5-5. Because nodes embed the transmission duration in headers and trailers, node v can detect that there was a concurrent transmission from node x by checking for overlapping transmissions from the headers and trailers received around the time a frame from u is received. Note that the transmission duration helps identify overlapping transmissions even when the headers and trailers are sent as separate frames and are delivered with a lag by the hardware. Also note that if v can never decode x's header, the conflict remains undetected. However, our evaluation in Section 5.5.4 shows that such cases are rare.

To measure the change in channel quality due to a concurrent transmission, node v computes the frame loss rate from u using the sequence number embedded in the headers and trailers. Node v keeps two different estimates of the frame loss rate—one when it thinks node u sent a frame in the absence of any interference (i.e., it did not detect any headers or trailers from any other node), and one for when it thinks there was a concurrent transmission with sender x. (Node v maintains an estimate not just for interfering sender x but every other concurrent sender it hears.) To first approximation, node u gets twice as many transmission opportunities by transmitting concurrently with $x \rightarrow y$ as compared to deferring to $x \rightarrow y$. Therefore, as long as the frame delivery rate of $u \rightarrow v$ when transmitting concurrently with $x \rightarrow y$ is at least half of that when u transmits by deferring to x, the throughput of $u \rightarrow v$ is higher when transmitting concurrently. By the definition of a conflict in Section 5.2.1, node v adds the entry $(u \rightarrow v, x)$ to its interferer list when it finds that the frame delivery rate of $u \rightarrow v$ in the presence of $x \rightarrow y$ is less than half of the frame delivery rate of $u \rightarrow v$. This interferer list is later broadcast by v and is used to populate the defer tables at nodes u and x.

Transmit decision. Every node in CMAP maintains a list of ongoing transmissions by decoding headers when it is idle. When a node receives a packet to transmit from the higher layers, the sender checks if the proposed transmission conflicts with any of the ongoing

transmissions using the transmit decision algorithm as described in Section 5.2.3. If there is no conflict, the node transmits the packet. If the conflict map does not allow a transmission, the node waits for a small duration $t_{deferwait}$ after the end of the current transmission and checks the list of ongoing transmissions for conflicts again. The transmission duration embedded in headers and trailers indicates how long a deferring node should wait.

Windowed ACKs. When exposed terminals transmit concurrently, the ACKs from the two receivers may interfere at the senders. The sender may then mistake the absence of an ACK for the loss of a frame and retransmit, losing out on the benefits of concurrency. Therefore, the CMAP retransmission protocol uses a window, unlike current wireless LAN link layers that use a stop-and-wait retransmission protocol (i.e., a window size of 1). The ACKs sent by receivers are cumulative and contain a bitmap indicating which frames in the window have been received. The main benefit of the window mechanism is to avoid spurious retransmissions when only the ACK (and not the data frame) gets lost, thereby making the retransmission protocol resilient to ACK losses.

After transmitting a packet, a sender waits for an ACK from the receiver for up to a duration $t_{ackwait}$. If the ACK does not arrive in this duration, the sender does not immediately retransmit the packet. It instead transmits a *send window* of up to N_{window} unacknowledged packets. We use $N_{window} = 8$ packets to tolerate a significantly higher loss rate on ACKs than on the data packets.

When the number of unacknowledged packets at a sender reaches the maximum N_{window}, the sender times out for a random time chosen between the minimum timeout τ_{min} and maximum timeout τ_{max} before retransmitting each unacknowledged packet in its window in sequence. Because the absence of an ACK for the entire window may indicate extreme interference at either the sender or the receiver, τ_{max} should be at least as long as the time taken to transmit an entire window's worth of packets in order to allow the interfering transmission at the destination to complete. We pick $\tau_{max} = \frac{N_{window}\ (bits)}{link\ speed\ (bits/s)}$ and $\tau_{min} = \frac{\tau_{max}}{2}$.

Backoff policy. CMAP, like most contention-based MAC protocols, uses a backoff algorithm to space out transmissions in response to contention on the channel. Every node maintains a contention window CW, and waits for a random duration in $(0, CW)$ at the end of a transmission before transmitting the next packet. Because CMAP uses cumulative ACKs, senders in CMAP, unlike 802.11 senders, do not update CW every time a transmission fails to elicit an ACK. Instead, senders compute the loss rate over a window of frames using the cumulative ACKs, and back off when this loss rate exceeds a threshold. CMAP senders update their contention backoff window CW upon receiving an ACK as follows. If the loss rate computed is below a threshold (0.5 in our implementation; we found in our evaluation that CMAP's performance was not sensitive to the choice of the threshold), the sender resets its CW to CW_{min}. If the loss rate is above the threshold, the sender doubles CW, up to a maximum CW_{max}. Note that the conflict map protocol may not work when the conflicting senders cannot hear each other (the hidden terminal problem) or when the receiver is not in the hearing range of, but experiences interference from, a far-away interferer. Nodes also experience transient losses before the feedback from receivers propagates to the defer tables of the senders. The backoff algorithm helps to slow down the senders

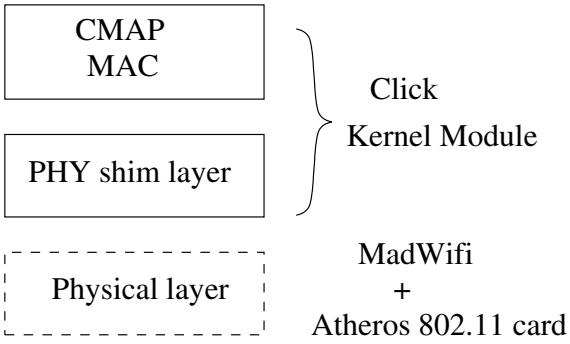


Figure 5-6: Architecture of the CMAP prototype on commodity 802.11 hardware.

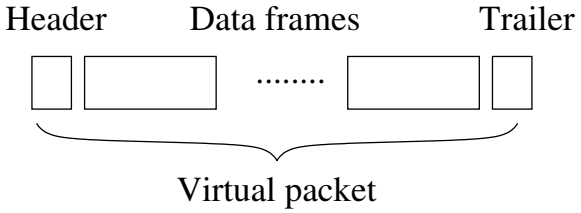


Figure 5-7: The concept of a "virtual packet" in the CMAP implementation comprising of "header" and "trailer" frames before and after a train of data frames.

and limit losses in such cases.

5.3.2 Implementation

We have implemented CMAP using the Click [36] router kernel module on Linux, and commodity 802.11 hardware driven by MadWifi [40], as shown in Figure 5-6. To control channel access and retransmissions from the CMAP kernel module, we disabled carrier sense, link-layer ACKs, retransmissions, and 802.11 backoff in the wireless card. All the nodes run in the promiscuous ("monitor") mode of the MadWifi driver. Our CMAP MAC implementation includes the logic to perform concurrent transmissions using the conflict map, as well as mechanisms to build and maintain the defer tables. We also implement a "shim" layer that transmits separate header and trailer frames immediately before and after a data transmission respectively. We will refer to the header, data, and trailer frames together as a "virtual packet". The shim is implemented in Click and is located between the CMAP MAC and physical layers.

Despite a kernel-level implementation, the communication latency between the CMAP code and the 802.11 PHY is significant in our implementation. For example, in a typical experiment, this latency was observed to be between 0.5 and 2 milliseconds for about 90% of the data packets, and between 2 and 5 milliseconds for the rest. In comparison, it takes only about 2 ms to transmit a 1400-byte packet at the lowest data rate of 6 Mbps in 802.11a. This communication latency poses two challenges. First, the gap between the end of a data transmission and arrival of the corresponding ACK is high in our implementation, leading unnecessary time wasted waiting for an ACK. Second, the delay may prevent senders from

Figure 5-8: Format of a transmission frame in SoftCMAP. A PHY synchronization postamble and a trailer are appended to the end of the data payload. The MAC header and trailer are protected by a separate CRC.

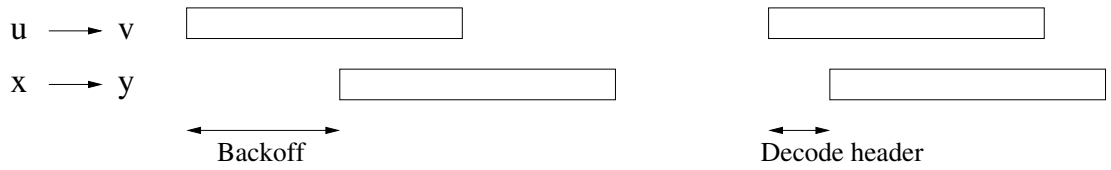
processing the received headers of conflicting transmissions before transmitting data.

To overcome these limitations, our implementation sends N_{vpkt} data frames destined to the same node between a header and trailer in one virtual packet, as shown in Figure 5-7. This approach effectively amortizes the cost of waiting for an ACK over multiple data frames, and allows senders to react in a timely manner to concurrent transmissions. In this implementation, defer and backoff decisions are made once per virtual packet; once a decision to transmit a virtual packet is made, the header, trailer, and all the data frames are sent without any gap in between. The receiver sends an ACK after receiving the trailer of a virtual packet; the bitmap contained in the ACK acknowledges the receipt of individual data frames within a virtual packet.

We now discuss the implementation choices for the various design parameters of CMAP. Our implementation uses $t_{deferwait} = t_{ackwait} = 5$ ms to accommodate the propagation delay between the MAC and physical layers. We use $N_{vpkt} = 32$ in our implementation because such a CMAP implementation has comparable performance to the commodity 802.11 protocol—the single link throughput of CMAP at 6 Mbps is 5.04 Mbps while that of 802.11 with link-layer ACKs is 5.07 Mbps—enabling a fair comparison of CMAP and 802.11. The send window of unacknowledged packets is set to $N_{window} = 8$ virtual packets, or 256 data frames. The contention window parameters CW_{min} and CW_{max} are set to the corresponding 802.11 values scaled by N_{vpkt}—5 ms and 320 ms respectively.

5.4 The SoftCMAP Protocol

The CMAP protocol described above has a few shortcomings arising from constraining the design to run on commodity 802.11 hardware. Because CMAP uses frame loss rates to infer conflicts, the conflict map requires many frame transmissions to converge and nodes require an infeasibly large number of measurements to populate the rate table. CMAP emulates the streaming PHY by replicating the header and trailer in separate frames, adding additional overhead to the data transmission. This section describes the SoftCMAP protocol, a clean-slate design of a conflict map-based MAC protocol that is unconstrained by the capabilities of today's commodity hardware. SoftCMAP inter-operates with CSMA and improves the number of successful concurrent transmissions in the network using the two complementary ideas of conflict maps and the streaming SoftPHY interface.



(a) Unaligned transmissions (b) Aligned transmissions

Figure 5-9: Two ways of performing concurrent transmissions in conflict map protocols: (a) unaligned transmissions where the start of the transmission is dictated by when the MAC backoff timer expires, and (b) aligned concurrent transmission where a non-conflicting transmission triggers a concurrent transmission.

5.4.1 Design Overview

PHY interface. SoftCMAP runs on the streaming PHY interface. The packet payload is followed by a synchronization "postamble" and PHY header, followed by a MAC "trailer" containing the sender and receiver MAC identities at the end of the packet. Our proposed packet format is shown in Figure 5-8. By synchronizing with the postamble and decoding the trailer, nodes know what transmission just finished even if they missed the header of the corresponding transmission. Adding the postamble and trailer does not change the design of the receiver dataflow in any significant manner. A receiver synchronizes with the postamble and decodes the trailer as if it were decoding the preamble and MAC header at the start of the transmission. The receiver prepares to decode the payload after decoding a header; it simply resets the receiver pipeline and waits for a new packet on decoding a trailer. Because the conflict map mechanisms depend on the correct reception of headers and trailers even in the presence of interference, the MAC headers and trailers are always sent at the lowest bit rate. While this design choice increases the relative overhead of the header, especially when the payload is sent at a high bit rate, we believe that the performance gains of SoftCMAP justify this small overhead.

Note that our postamble synchronization is different from the more complicated scheme proposed in [26]. The earlier work proposed adding a trailer followed by the synchronization postamble at the end of the packet. With this design, receivers maintain a buffer of received samples always. When a receiver synchronizes with a postamble, it runs the buffered samples backwards through the receiver toolchain to decode the trailer and as much of the packet as it can. This way, a receiver that missed the preamble of a packet can still decode parts of the packet using the postamble. While the receiver is decoding the buffered samples this way, it must also detect and decode preambles of new transmissions. Therefore, such receivers must be capable of decoding two transmissions in parallel. As such, the receiver PHY architecture to implement such postamble synchronization is more complicated and requires significant changes to existing PHY designs. While the goal of the earlier postamble design was to enable decoding of the entire packet, our goal is more modest—to just recover header information from the completed transmission—and requires fewer changes to existing receiver PHY architectures.

Mechanism of concurrent transmissions. Recall that the time at which a packet is handed off to the PHY hardware for transmission in CMAP is decided by the time of arrival of the packet into the queue and the backoff timer at the MAC layer. Senders wait for the backoff timer on a packet to expire, then check what other transmissions are ongoing at the time the packet is ready for transmission, and decide to transmit if there are no conflicting transmissions. However, the benefits of concurrency are not fully exploited in this scheme because the non-conflicting transmissions start at different times and overlap over the air only for a fraction of their airtime. SoftCMAP adopts a different approach that increases throughput by maximizing the overlap of the two non-conflicting transmissions. When a SoftCMAP node decodes the header over the air and realizes that the transmission does not conflict with its next transmission at the head of the queue, the node preempts its backoff timers and *opportunistically* transmits a packet immediately so that the two non-conflicting transmissions align completely. That is, the first non-conflicting transmission *triggers* the second transmission. The SoftCMAP protocol performs such *aligned concurrent transmissions*, whereas the CMAP protocol performs *unaligned transmissions*. The two types of concurrent transmissions are illustrated in Figure 5-9.

A key challenge in implementing these aligned concurrent transmissions is the turnaround time in the PHY from decoding a header to switching the hardware to transmit mode and sending a packet. We will later show that rapidly switching from receive mode to transmit mode can be accomplished in a hardware MAC implementation. Another challenge that must be overcome is that multiple senders can decode a non-conflicting sender's header and decide to transmit at the same time, leading to more than two concurrent transmissions and potential losses. One of the main functions of MAC backoff timers is to jitter the start times of transmissions so that nodes have enough time to sense each other and make informed channel access decisions. However, by transmitting opportunistically before the backoff timers expire, nodes lose this benefit of the backoff mechanism.

To avoid this problem, we propose that a sender that sends a packet must explicitly *designate* a non-conflicting sender for a concurrent transmission. Consider a pair of non-conflicting transmissions $u \rightarrow v$ and $x \rightarrow y$. Suppose both u and x have packets to transmit and are aware that the other exposed sender has packets to send to a non-conflicting destination (we will soon describe how the nodes can learn of this information). Now suppose u's MAC timer expires and it gets ready to send a packet. We will call u the *primary sender*. Node u now designates node x to be its *secondary* sender in its MAC header. Now when node x decodes u's header, it alone starts a concurrent transmission with u, thereby gaining the benefit of increased concurrency while avoiding the risk of multiple senders starting opportunistic transmissions.

Let us examine the detailed timeline of this aligned concurrent transmission, as shown in Figure 5-10. Let T_h be the time taken by node u to transmit the PHY synchronization preamble, PHY header and the MAC header over the air, measured from when node u starts transmission. Let T_Δ be the time between u's header being sent over the air and node x initiating its concurrent transmission. This delay includes the (negligible) propagation delay over the air between u and x, the delay in decoding the MAC header at x and the time needed by the PHY at x to switch from receive to transmit mode. The two concurrent transmissions are now jittered by a duration of $T_h + T_\Delta$. Assuming equal transmission times for both the packets, the secondary transmission from node x finishes $T_h + T_\Delta$ after

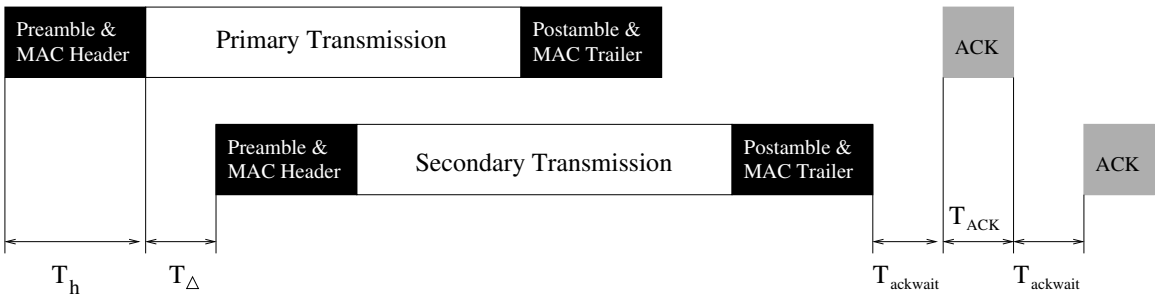


Figure 5-10: Timeline of the data and ACK transmissions in an aligned concurrent transmission, where a primary transmission triggers the second concurrent transmission.

the primary transmission. The primary sender and receiver try to decode the trailer from the secondary transmission when possible.

Note that even through u and x are exposed terminals, the ACKs from v and y can still collide at both the senders because v and y may not be exposed senders. Therefore, the ACKs from the two transmissions must also be jittered slightly in order not to collide at the senders. Suppose the receiver normally starts the transmission of an ACK after a duration $T_{ackwait}$ after the medium becomes free, and the transmit duration of the ACK is T_{ACK}. Then the primary receiver v should wait for $T_h + T_\Delta + T_{ackwait}$ to start the ACK transmission, in order to allow for node x's transmission to finish. Node y should wait for a duration $2T_{ackwait} + T_{ACK}$ to begin its ACK transmission, to allow for the primary ACK to be received successfully. Note that the values of all the timing parameters except T_Δ (i.e., the values of T_h, $T_{ackwait}$, and T_{ACK}) are fixed by the specifications of the MAC and the PHY. For example, the wait time before sending the ACK is 16 μs in 802.11a. Only T_Δ depends on the specific implementation of SoftCMAP. We will measure T_Δ for an FPGA-based implementation in Section 5.4.3.

Identifying backlogged flows. We define a flow at the MAC as a set of MAC frames between the same sender and receiver. We define backlogged flows as the set of flows that are awaiting a transmission opportunity at any instant. If nodes use per-destination MAC queues, backlogged flows include all flows with packets outstanding in the MAC queues. If the MAC uses only a single FIFO queue for all destinations and transmits packets only from the head of the queue, then only the flows at the heads of the MAC queues will be considered backlogged. The number of MAC-layer flows is often much smaller than the number of TCP or UDP flows, and depends on the routing topology of the network.

Let us revisit the above example where node u designates node x to be its secondary sender. Assuming realistic traffic patterns where flows are not always backlogged, how does u know that x has packets from a non-conflicting flow in its queue just when it is about to send a packet? We solve this problem by requiring that SoftCMAP nodes include the MAC queue length of a backlogged flow in the headers and trailers of the packets of that flow, so as to signal their intention of transmitting concurrently with other exposed senders. In fact, it is sufficient for nodes to signal if they have *any* outstanding packets in a flow besides the packet being transmitted, and the exact queue length does not matter.

Every node decodes headers and trailers on the air when it is idle, and updates a list of backlogged flows. Because a typical radio neighborhood may not have more than a few tens of outstanding flows at any time, a linked list is a suitable data structure to store this information.

Normalizing packet transmission time. To maximize the extent of overlap of non-conflicting transmissions, the packets from both the senders must have approximately equal transmission durations. However, even if all senders use the same frame size, the transmission durations can be widely different at different bit rates. A more serious problem with senders having different transmission durations at different bit rates is that frames sent at lower bit rates take up more time on the medium, crowding out frames at higher bit rates and lowering the aggregate throughput of the system. To address this inefficiency, the MAC should normalize the packet transmission times of the contending senders at all bit rates and provide all the contenders with equal airtime [63]. This notion of fairness is called time-based fairness, and can be achieved by either combining or fragmenting individual frames of a flow based on the current bit rate, such that all frames have approximately the same transmission duration at all bit rates.

Senders in SoftCMAP repacketize MAC frames to normalize the transmission duration to a network-wide constant whenever possible. This mechanism increases throughput for two reasons. First, it increases the amount of data sent at higher bit rates when compared to using just equal packet sizes at all the nodes. Second, it increases the overlap duration of aligned concurrent transmissions, irrespective of the transmit bit rates of the two concurrent senders.

Building the conflict map and rate tables. SoftCMAP nodes use the channel BER computed from SoftPHY hints during a transmission to infer conflicts, as well as pick appropriate bit rates for all the transmissions. In the case of aligned concurrent transmissions, this BER can be approximately computed by each receiver as the channel BER over the entire received frame, because most part of the frame is a concurrent transmission. Let us revisit the example of node u transmitting to v in the presence of $x \rightarrow y$. Let the rate table entry at u corresponding to transmitting to v without interference be r_1, and the entry corresponding to transmitting concurrently with x be r_2. Initially, r_2 is set equal to r_1, and the interferer lists and defer tables at the nodes do not indicate a conflict between the two transmissions.

Soon, node u and x will update their list of backlogged flows and transmit concurrently with each other. Node v computes the BER during one such concurrent transmission with $x \rightarrow y$ and sends the BER feedback to node u. Node u then uses this feedback to update the rate table entry r_2 corresponding to destination v and concurrent sender x as per the SoftRate protocol. Now, node u gets about twice the airtime by sending concurrently with node x, because u gets to transmit every time either u or x wins the contention on the channel. So, as long as $r_2 \geq \frac{r_1}{2}$, node u gets a higher throughput by transmitting concurrently. But if $r_2 < \frac{r_1}{2}$, the transmissions $u \rightarrow v$ and $x \rightarrow y$ conflict, at which point node u removes the entry (v, x, r_2) from the rate table and adds the entry $(u \rightarrow v, x)$ to its interferer list. This interferer list is then distributed to all nodes in periodic updates from node u, and is

used to subsequently update the defer tables at node u itself and at the interfering node x, as described in Section 5.2.2.

Note that the BER from SoftPHY hints is a particularly good feedback in SoftCMAP, because nodes require only a small number of concurrent transmissions to populate the conflict map.

Inter-operation with CSMA. Note that one sees throughput gains with SoftCMAP only for long flows, because the overhead of discovering the conflict map is amortized over many concurrent transmissions. The idea of overlapping aligned transmissions is also not useful for flows with small packets because the overhead of waiting for a duration $T_h + T_\Delta$ before starting a concurrent transmission is justified only if the packet transmission time is much greater than $T_h + T_\Delta$. Therefore, the SoftCMAP protocol inter-operates with CSMA. Nodes that do not wish to participate in the SoftCMAP protocol simply set a queue length of zero in their headers, signaling to other nodes that they do not wish to be part of any concurrent transmissions. Nodes that opt in can gain throughput benefits by opportunistically transmitting with other senders. Using only long flows also ensures that the MAC has enough packets in each flow to repacketize frames and normalize the transmission durations.

5.4.2 Design Details

We now describe the packet format and the sender and receiver algorithms in SoftCMAP.

Packet format. SoftCMAP adds the following additional fields to the 802.11 MAC header. Headers and trailers are in the same format.

1. A 1-bit flag to indicate if it is the header or the trailer. Headers and trailers are identical but for this one bit.

2. A 1-bit flag indicating if a transmission is the primary or secondary transmission in a primary-secondary pair.

3. A 48-bit MAC identifier of the other sender in case the transmission is part of a primary-secondary pair. That is, the primary sender uses this field to place the secondary sender's identifier and vice versa. If a transmission is not part of a primary-secondary pair, then this field is zeroed out.

4. A 1-bit flag indicating if the node has outstanding packets corresponding to that flow in its MAC queue. To avoid triggering concurrent transmissions if a node has just one or two packets of that flow, a node can set the outstanding packets flag only if the number of packets in its queue exceeds a certain threshold, say, 5. Nodes can set this flag to indicate no outstanding packets if they want to opt out of the SoftCMAP protocol and follow plain CSMA.

5. A 32-bit CRC covering the information in the header or the trailer to enable error detection on the header information before frame reception completes.

Therefore, SoftCMAP adds 12 extra bytes to the MAC header: 6 bytes to transmit the identity of the secondary or primary sender, 4 bytes for a separate CRC over the MAC header, and 2 extra bytes for other information such as the flag to distinguish headers and trailers, the flag to identify the primary and secondary transmissions, and the flag to indicate a backlogged flow. The payload of every data packet is followed by a synchronization postamble and a trailer. A data transmission is always followed by a link-layer feedback frame. The feedback frame is the link-layer ACK in case of a successful packet transmission. Otherwise, the feedback frame is the same as the ACK, except for a different frame type in the header indicating that the packet reception failed. We add one additional field to the frame: a 4-byte feedback of channel BER. Note that we will use the term ACK and feedback frame interchangeably in the rest of the discussion.

Data structures at every node. Every node maintains a list of backlogged flows, populated from overhearing headers and trailers when idle. Every node periodically (say, every few seconds) sends and receives special control packets containing its current interferer list. Every node that receives an interferer list from another node updates its defer tables to reflect the latest conflict information.

At the sender. When a sender's backoff timer expires and it has to send a packet, the node checks if the carrier is idle. If the medium is idle, the node checks the list of backlogged flows and the conflict map to find a non-conflicting backlogged sender. That is, the node runs the checks described in Section 5.2.3 for every sender-receiver pair in the list of backlogged flows. If the node can find a non-conflicting sender s, the node sets its primary flag in the header, designates sender s as its secondary, and transmits the packet. The transmit bit rate is picked from the rate table assuming the node will be transmitting concurrently with the secondary sender s. Note that this bit rate may be different from the rate the sender would have used had it not designated any node as its secondary. When a node starts transmission as a secondary sender, the node sets the secondary flag in its header, places the identity of the primary sender in its header, chooses an appropriate rate for the concurrent transmission, and proceeds to transmit.

At the end of the transmission, the primary sender switches its radio to receive mode, waits for the end of the secondary transmission by carrier sensing the medium, and tries to decode the trailer of the secondary transmission if possible. Decoding the secondary sender's trailer lets the primary update its list of backlogged flows. For example, if the secondary sender s no longer has any outstanding packets in the flow, the primary sender can update this information and not pick node s as its secondary the next time it makes a transmission. If the primary senses the medium to be idle after its transmission, it may assume that the secondary node did not transmit either because it did not have any packets or because the defer tables were inconsistent at both the nodes, and removes the secondary node from its list of backlogged nodes until it receives further headers or trailers from that node. If the primary could not decode the secondary's trailer due to poor signal quality, the nodes lose out on potential opportunities for successful concurrent transmissions with each other; the correctness of the protocol is not affected. The primary may not see the secondary's trailer if the secondary transmission is much shorter than the primary trans-

mission; normalizing frame transmission durations avoids this possibility.

Every sender implements an exponential backoff algorithm, much like CSMA. Nodes maintain a contention backoff window CW. When there is a packet to send, nodes wait for a random backoff duration in $(0, CW)$ before attempting to transmit the packet. CW is set to an initial value of CW_{min} at the start of time. Upon a packet loss, the value of CW is doubled up to a maximum CW_{max}. Upon a successful transmission, CW is reset to CW_{min}. CW_{min} and CW_{max} are chosen to mirror the corresponding 802.11 values. The retransmission policy is also similar to 802.11. After transmitting a packet, a sender waits for an ACK from the receiver (for up to a duration that depends on whether the sender is the primary or secondary sender). If the sender does not receive a feedback frame, or if the feedback frame does not indicate a successful reception, then the sender retransmits the packet for a fixed number of times. That said, SoftCMAP can use any other link-layer error recovery mechanism as well.

At the receiver. Receivers always look to decode headers and trailers of all transmissions over the air. One of three things can happen when a node decodes a header:

1. When a node decodes the header of a transmission where it is the receiver, it proceeds to decode the rest of the packet. At the end of receiving a packet, the node begins to carrier sense the medium until it becomes free (possibly from a concurrent transmission). During this time, the receiver tries to decode any new headers or trailers as usual. After the medium becomes free, the receiver turns around to send an ACK after waiting for an interval of time $T_{ackwait}$ or $T_{ACK} + 2T_{ackwait}$ depending on whether the reception was the primary transmission or the secondary. The ACK contains the channel BER estimated from SoftPHY hints during the transmission.

2. If the node is not the receiver, then the MAC instructs the PHY to *abort* the reception, i.e., to flush the state of all the modules in the receiver pipeline of the PHY and start looking for new preambles. This abort mechanism is needed because a concurrent transmission might start later on for which the receiver is the intended destination, and the receiver would not want to miss that preamble.

3. If the receiver is not the intended destination in the header just decoded, but is designated as the secondary sender, then the receiver aborts the reception and switches to transmit mode in order to transmit as a secondary sender.

5.4.3 Implementation

We evaluate the steady state throughput of SoftCMAP using trace-driven simulations with SoftPHY-capable GNURadio prototypes (Section 3.3). We evaluate the conflict map mechanisms that are common to both protocols using the CMAP prototype alone, because the CMAP prototype on commodity hardware runs at a much higher and more realistic bandwidth than software radios. We now describe how we compute the aggregate throughput of a set of four nodes (two sender-receiver pairs) performing aligned concurrent transmissions with SoftCMAP, given the channel error rate observed by the two senders during a

concurrent transmission. This computation will be used in the trace-driven simulations of SoftCMAP in Section 5.5.

Consider a sender (primary or secondary) performing an aligned concurrent transmission. Recall that SoftCMAP equalizes the transmission time of frames at all bit rates by repacketizing data. Assume that payload frames are packetized to fit in transmission slots of size $T_{payload}$ = 1867 μs, which is the time taken to send a 1400-byte packet at 6 Mbps. For example, the MAC combines two 1400-byte frames when transmitting at 12 Mbps and so on. Let the total transmission time of a payload frame with the overhead of the headers, trailers, and other SoftCMAP mechanisms be T_{total}. The transmit time T_{total} has the following components, as shown in Figure 5-10.

1. Time taken to transmit the header, MAC payload, and the trailer of a frame. If T_h is the time taken to transmit the PHY synchronization preamble (or postamble) and the MAC header (or trailer), then the time taken to transmit a frame is $2T_h + T_{payload}$.

2. The jitter between the transmissions $T_h + T_\Delta$. Both the primary and secondary transmissions incur this delay: the secondary sender stays idle for this duration before its transmission begins, and the primary waits for this duration after its transmission for the medium to become idle.

3. The time spent waiting for an ACK. Each sender spends a time $2T_{ackwait} + 2T_{ACK}$ for both the ACK transmissions to complete.

Adding up the above numbers, the total time taken to send a data frame is given by:

$$T_{total} = 2T_h + T_{payload} + T_h + T_\Delta + 2T_{ackwait} + 2T_{ACK} \tag{5.1}$$

If the loss rate of the sender in the presence of the concurrent transmission is l, then the expected time to send a frame assuming frame-level retransmissions is $\frac{T_{total}}{1-l}$. (We are ignoring the impact of backoff wait time.) If the bit rate of the transmission is R, then the sender effectively transfers $RT_{payload}$ units of data in this time. Therefore the effective throughput of the sender is:

$$\text{SoftCMAP throughput} = \frac{RT_{payload}(1-l)}{T_{total}} \tag{5.2}$$

The throughput of the second concurrent sender can also be computed similarly, using its value of the perceived loss rate and its transmit bit rate. The aggregate throughput of the two transmissions is obtained by adding the throughput of the two concurrent senders.

All the timing parameters except T_Δ in Equation 5.1 are fixed by the specification. In our simulation of SoftCMAP, we use 802.11a-like values for the timing parameters:

1. The preamble is 16 μs long and the PHY header is one OFDM symbol or 4 μs. The 802.11 MAC header is 24 bytes long, and the SoftCMAP protocol requires an additional 12 bytes in the header (Section 5.4.2). At the base rate of 6 Mbps, the 36-byte MAC header transmission takes 48 μs. Therefore, the total time taken to transmit the preamble and MAC header is T_h = 68 μs.

2. $T_{ackwait}$ is the SIFS interval (16 μs).

3. The size of the ACK frame is 14 bytes in 802.11a. With 4 bytes of BER feedback, the ACK frame is 18 bytes long in SoftCMAP. At the lowest rate of 6 Mbps, the ACK transmission takes 44 μ—20 μs for the preamble and PHY header, and 24 μs for the actual ACK frame.

Below we describe how we estimate the value of T_Δ using an FPGA-based implementation of the mechanism of concurrent transmissions. Once the value of T_Δ is known, we can compute the transmission time of any frame given its payload size using Equation 5.1, and compute the aggregate steady state throughput of SoftCMAP with concurrent transmissions using Equation 5.2.

Measuring T_Δ. We see from Figure 5-10 that T_Δ is the time from when the header of the primary transmission is sent over the air to when the secondary transmission starts. To measure T_Δ, we start with the Airblue 802.11 PHY implementation that was modified to provide SoftPHY hints (Section 3.3.2). The PHY in Airblue passes up decoded data and SoftPHY hints to the MAC at the granularity of bytes in the same clock cycle that they are decoded, conveniently realizing our streaming PHY abstraction. We then modify the MAC to start a concurrent transmission after receiving the primary sender's header, and measure the value of T_Δ in the FPGA implementation.

The delay T_Δ has two components. First, the secondary sender's PHY must finish decoding the bits corresponding to the MAC header and stream them to the MAC. Second, the secondary sender's MAC must start a concurrent transmission in response to the primary's MAC header. We describe each step in more detail.

1. *Decoding the MAC header.* We modify the Airblue MAC and PHY to transmit the MAC header and trailer at the base rate of 6 Mbps, unlike in 802.11 where it is transmitted at the bit rate of the payload. The receiver PHY starts decoding the PHY symbols corresponding to the MAC header as soon as they are received from the RF front-end, and exports the bits coming out of the pipeline to the MAC. We saw in Section 3.3.2 that the processing latency of the receiver PHY pipeline in Airblue is 15.52 μs when computing SoftPHY hints. Therefore, the last bit of the MAC header reaches the receiver MAC 15.52 μs after it is transmitted over the air.

   We found that besides improving the reliability of the control information, transmitting the header at the lowest bit rate allows the PHY to pass the header up to the MAC sooner. When the 802.11 MAC header is transmitted at the payload of the packet, the 802.11 receiver PHY must first decode the PHY header that specifies the bit rate of the MAC header, and then reconfigure the pipeline to operate at the (possibly different) bit rate of the MAC header. This reconfiguration of the PHY adds a delay before the PHY symbols corresponding to the MAC header can be decoded and streamed up to the MAC, unlike in SoftCMAP.

2. *Initiating a concurrent transmission.* When a node receives the MAC header corresponding to a primary transmission from the PHY, it checks if it is the designated

secondary secondary. If it is, the node picks a frame from a non-conflicting flow to transmit, and readies the PHY for a transmission. Note that the PHY of the secondary sender would have been receiving the primary transmission at this point, and the various PHY modules will have state from that reception. If the secondary sender simply transmits without clearing this state, it may lead to a corruption of subsequent packet receptions. Therefore, the secondary sender's MAC first issues a request to *abort* the reception of the primary transmission and clear all state from the receiver's pipeline. The MAC sends an abort request to the head of the receiver pipeline (RX Controller in Figure 3-4). The RX controller then inserts a special "abort token" through the pipeline and stops forwarding data along the pipeline. The abort token in our case is simply a new zero-length packet header. Every module that receives the token prepares for receiving a zero-length packet, and flushes the state of the packet it was receiving before. When the abort token reaches the end of the pipeline, the PHY sends a response to the MAC to indicate the completion of the abort. The secondary sender's MAC then initiates the secondary transmission. In our implementation, the time delay between the secondary sender's MAC receiving the MAC header of the primary transmission and the secondary sender starting a concurrent transmission was found to be 8.4 μs, dominated by the time taken to flush the state of the primary reception.

Therefore, a secondary sender can initiate a secondary transmission with a delay of T_Δ = 15.52 + 8.4 $\approx$ 24 μs after the transmission of the primary transmission's MAC header. Observe that the lower the value of T_Δ, the higher the throughput gains of SoftCMAP. The delay of 24 μs we observe in our implementation is small enough compared to typical packet transmission times (hundreds to thousands of microseconds) and does not impose a significant overhead on SoftCMAP throughput. Also, a more optimized implementation of the SoftCMAP mechanisms in hardware can shrink this delay even further.

5.5 Evaluation

In this section, we describe the evaluation of the two conflict map-based protocols, SoftCMAP and CMAP. Because of our implementation of CMAP over commodity hardware, we are able to evaluate CMAP in live experiments. We evaluate SoftCMAP by performing trace-driven simulation using traces from software radio experiments. Below are the main findings of our experiments.

1. In experiments with pairs of transmissions, the conflict map mechanism can accurately identify the conflicting transmissions from the exposed terminal cases, and selectively enables concurrent transmissions in the exposed terminal cases alone.

2. Our experiments with pairs of exposed terminals show that the protocols achieve a median $1.7\times$ gain over CSMA, even with the overheads of the various conflict map mechanisms.

Figure 5-11: A map of our 50-node indoor 802.11a testbed for the evaluation of CMAP.

3. Experiments in larger topologies with more than two transmissions show that harnessing exposed terminals can lead to up to a 50% improvement in throughput in common network topologies.

5.5.1 Method

Evaluation of CMAP. We evaluate CMAP over a 50-node indoor 802.11 testbed. Our 802.11 testbed consists of Soekris net4526 computers with a 133 MHz 486 processor running the 2.4.26 Linux kernel. The nodes are equipped with an 802.11a/b/g mini-PCI card based on the Atheros AR5212 chipset. The testbed nodes are located in one large floor of an office building as shown in Figure 5-11. Of the 2162 node pairs that have any connectivity whatsoever, approximately 68% links have a packet reception rate (PRR) less than 0.1, 12% have a PRR greater than 0.1 and less than 1, and 20% have a PRR of 1. Considering just the latter two types of links, the nodes in our testbed have a mean degree of 15.2 and a median degree of 17. We perform all our experiments in the 5 GHz 802.11a band which had negligible background traffic. Unless mentioned otherwise, all senders transmit 1400-byte data packets at the 6 Mbps bit-rate of 802.11a as fast as they can. Each run of an experiment lasts for 100 seconds and the aggregate throughput achieved is measured as the total number of non-duplicate data packets received per second at the designated receivers, computed over the last 60 seconds of the experiment.

We pick sender-receiver pairs in each experiment based on measurements of PRR and average signal strength between pairs of nodes at 6 Mbps and in the absence of any other concurrent transmission, obtained shortly before running the corresponding experiment. In all experiments below, we define two nodes a and b to be "in range" of each other if both the links $a \rightarrow b$ and $b \rightarrow a$ have a PRR above 0.2 and signal strength above the 10th

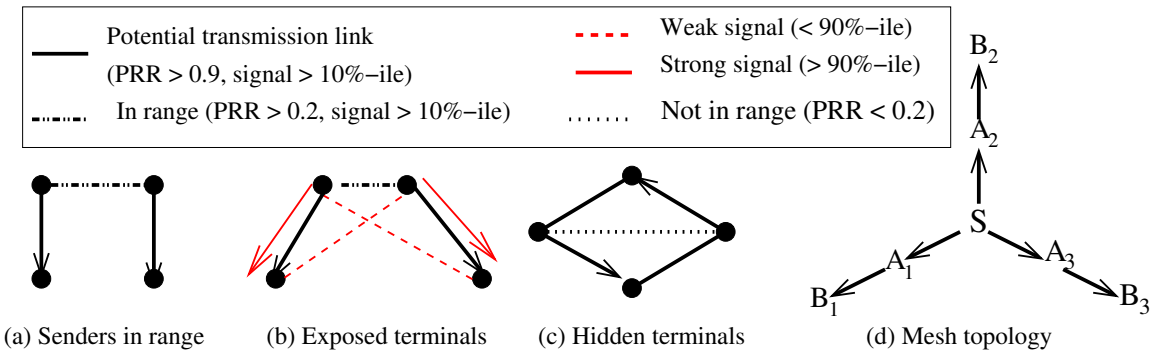


Figure 5-12: Constraints on topologies in CMAP experiments with (a) pairs of random senders (Section 5.5.2), (b) pairs of exposed terminals (Section 5.5.3), (c) pairs of hidden terminals (Section 5.5.4), and (c) a tree-based mesh network (Section 5.5.5). All links that are not annotated in the figure are unconstrained.

percentile of the signal strength of all links network-wide. We call a link $a \rightarrow b$ a "potential transmission link" if both the links $a \rightarrow b$ and $b \rightarrow a$ have a PRR above 0.9 and signal strength above the 10th percentile of the signal strength of all links network-wide; we pick only such links to send data in our experiments because any routing protocol running on this testbed typically selects such links. Note that while the PRR of a link alone could serve as a good metric to decide whether the link is a potential transmission link or not, we also use a low signal strength threshold to eliminate very weak links whose performance would degrade precipitously in the presence of other transmissions in the network. Figure 5-12 summarizes how we pick topologies for the various CMAP experiments in this section; the details will be described in the context of the individual experiments.

In our experiments, we compare CMAP to two flavors of 802.11: with carrier sense turned on and off. The 802.11 throughput is obtained by running the UDP flows on the wireless devices without using the CMAP mechanisms.

Evaluation of SoftCMAP. We evaluate SoftCMAP using trace-driven simulations, with traces collected from our SoftPHY-capable software radio prototype (Section 3.3.2). We collect packet traces and SoftPHY hints from experiments with four nodes configured as two sender-receiver pairs. Each pair of senders transmits packets at every combination of the 6 available bit rates (corresponding to the 802.11a rates of 6, 9, 12, 18, 24, and 36 Mbps), both one at a time and concurrently. The data in these traces was used to generate Figure 5-2 in Section 5.1. The information collected from the traces includes the frame loss rates of the senders for every possible combination of bit rates and concurrency, and the average channel BER computed from SoftPHY hints in each case.

In each experiment, we compute the optimal combination of bit rates of the two senders during concurrent transmissions using the rate selection algorithm of the SoftRate protocol. Figure 5-13 shows the per-frame average BER computed from SoftPHY hints (from frames in the traces that have overlapping concurrent transmissions) plotted against the actual BER of the frame. We see from the figure that SoftPHY hints can be used to estimate channel BER accurately during the concurrent transmissions, and therefore can be used to pick the

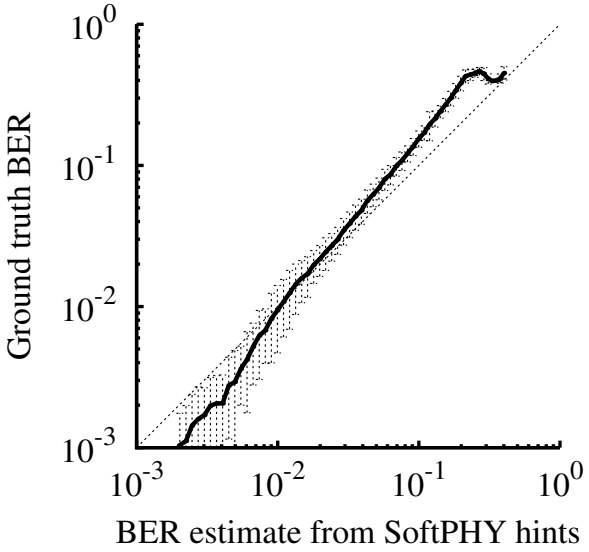


Figure 5-13: Per-frame average BER estimated by SoftPHY hints vs. the actual BER of the frame from experiments with concurrent transmissions using two software radio senders configured as exposed terminals.

best transmit bit rate combination of the two senders. We then use the experimental traces of concurrent transmissions at the optimal bit rates of the two senders, and compute the steady state throughput of SoftCMAP using the computation described in Section 5.4.3.

We compare SoftCMAP against 802.11 with CSMA for each pair of senders. The throughput of a transmission with CSMA is computed using the frame loss rates over 1400-byte frames, and adjusting it for 802.11 MAC timing overheads. For each sender-receiver pair, we use the maximum throughput achieved across all bit rates as the link throughput, because we assume that CSMA is running with a smart rate adaptation protocol that can converge to the best possible transmit bit rate for a single link. Given a pair of senders with single link throughputs of R_1 and R_2, we compute the aggregate throughput of the two transmissions with CSMA as the harmonic mean of the two throughputs ($\frac{2R_iR_2}{R_1+R_2}$), because CSMA gives equal transmission opportunities to each sender, resulting in packets at lower rates taking up more airtime. Because SoftCMAP repacketizes frames to give equal airtime to senders at different bit rates, we also simulate a flavor of CSMA with such time-based fairness to enable a fair comparison with SoftCMAP. We assume that CSMA with time-based fairness uses the same data frame sizes as SoftCMAP. If two senders have link throughputs R_1 and R_2, then the aggregate throughput under this variant of CSMA is $\frac{R_1+R_2}{2}$.

5.5.2 Identifying Exposed Terminals

We first run CMAP over 50 pairs of senders, chosen randomly from the testbed nodes, and verify if we can selectively enable concurrent transmissions between exposed terminals using the conflict map mechanism. In this experiment, we choose two sender-receiver pairs as shown in Figure 5-12(b): the two senders are in range of each other, and each sender-

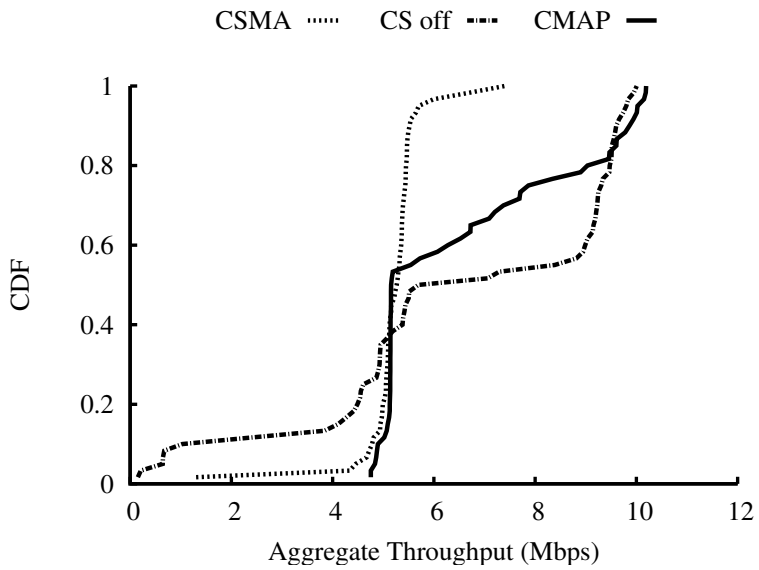


Figure 5-14: Aggregate throughput of two transmissions in an experiment with two senders in range of each other, comparing the performance of CMAP, CSMA, and CS off. The CDF is plotted over 50 experiments with different sets of nodes.

receiver pair is a potential transmission link. We impose no additional constraints on the signal strengths of the links. Note that some pairs of links could well be exposed terminals, and some interfering. The data from this experiment was used to generate Figure 5-1 in Section 5.1.

Figure 5-14 presents the distribution of throughput across all the pairs. On about 40% of the link pairs, simultaneous transfers were deleterious, evidenced by the worse performance of 802.11 with carrier sense disabled compared to 802.11 with carrier sense enabled. For these link pairs, CMAP correctly directs that each transmission defer to the other and tracks the performance of 802.11 with carrier sense on (5 Mbps). However, the link pairs on the right-hand side of the CDF are better off transmitting concurrently, because 802.11 with carrier sense and link-layer ACKs disabled offers a throughput improvement up to $2\times$. For these link pairs, CMAP correctly directs transmissions to occur simultaneously, and thus achieves roughly the same throughput improvements as 802.11 with carrier sense disabled. Note that we disabled link-layer ACKs as well along with carrier sense in order to generate the "CS off" curve above. Therefore the performance of the carrier sense disabled experiment does not reflect the ACK loss problem in exposed terminals, and is an upper bound on CMAP's performance. This experiment convinces us that our defer scheme correctly discriminates between interfering and non-interfering concurrent transmissions.

5.5.3 Performance Gains with Exposed Terminals

Having shown that the conflict map mechanism can correctly identify exposed terminals from conflicting transmissions, we now focus solely on exposed terminals to understand the performance benefits of using the conflict map mechanism. We present results with CMAP and SoftCMAP over exposed terminal configurations.

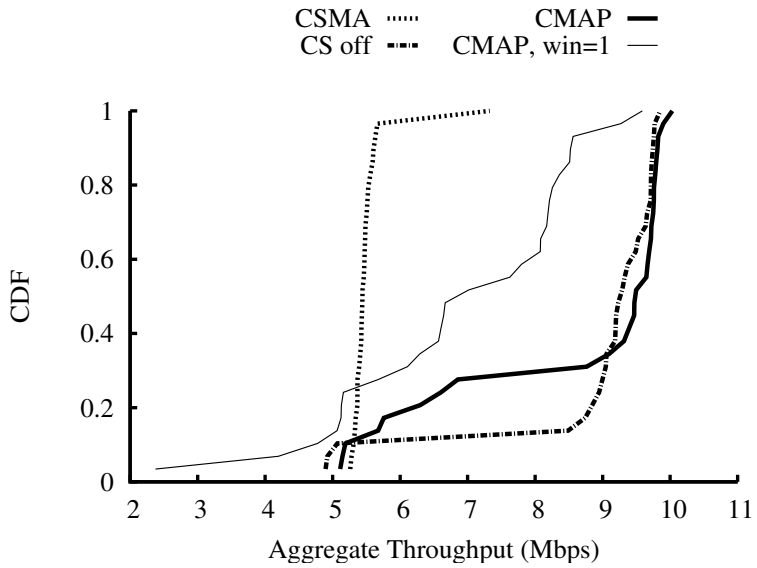


Figure 5-15: Aggregate throughput of two transmissions in an experiment with exposed terminals, comparing CMAP, CMAP without windowed ACKS, CS on and CS off. The CDF is plotted over 50 experiments with different sets of nodes.

Gains with CMAP. We first perform an experiment with pairs of exposed senders running the CMAP protocol. This experiment only measures the gains at the fixed bit rate of 6 Mbps because the CMAP protocol does not handle heterogeneous bit rates. We pick pairs of links from our 50-node testbed, as shown in Figure 5-12(a), such that: (i) the two senders are in range of each other, (ii) each sender-receiver pair is a potential transmission link (as per the definitions above), (iii) the signal strength from a sender to its corresponding receiver is strong (in the 90th percentile of signal strength of all links network-wide), and (iv) the signal strength between all other pairs of nodes is somewhat weak (below the 90th percentile threshold). The last two constraints ensure that the two transmissions do not interfere very much, most likely resulting in an exposed terminal situation.

Figure 5-15 presents the distribution of throughput across 50 exposed terminal configurations chosen at random from all possible configurations. With carrier sense enabled, we see that most link pairs achieve little more than the single-link rate of 5 Mbps, since most of the time, each sender defers its transmission to the other. With carrier sense and link-layer ACKs disabled (in order to avoid the penalty from ACK collisions), we see that 15% of the pairs at the bottom left of the graph are not in fact exposed terminals in the sense that the two transmissions do not simultaneously achieve their maximum throughput. Of the remaining 85% of pairs in this experiment, CMAP permits the two transmissions to proceed concurrently $\frac{70\%}{85\%} = 82\%$ of the time, resulting in a throughput improvement of approximately $2\times$ over 802.11 with carrier sense enabled. By carefully scrutinizing the experiment logs, we verified that the remaining 18% of pairs experienced many spurious retransmissions due to very high ACK loss rates that our windowed ACK scheme could not completely eliminate.

To quantify the benefits of our windowed retransmission protocol, we repeated the CMAP experiments with a window size of one virtual packet. We found that the me-

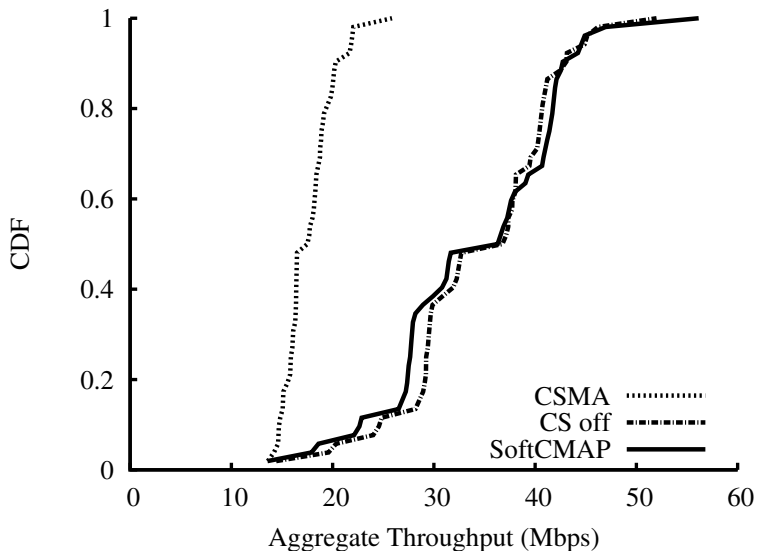


Figure 5-16: Aggregate throughput of two transmissions in trace-driven simulations with exposed terminals, comparing SoftCMAP, CSMA, and CS off. The CDF is plotted over 50 experiments with different sets of nodes.

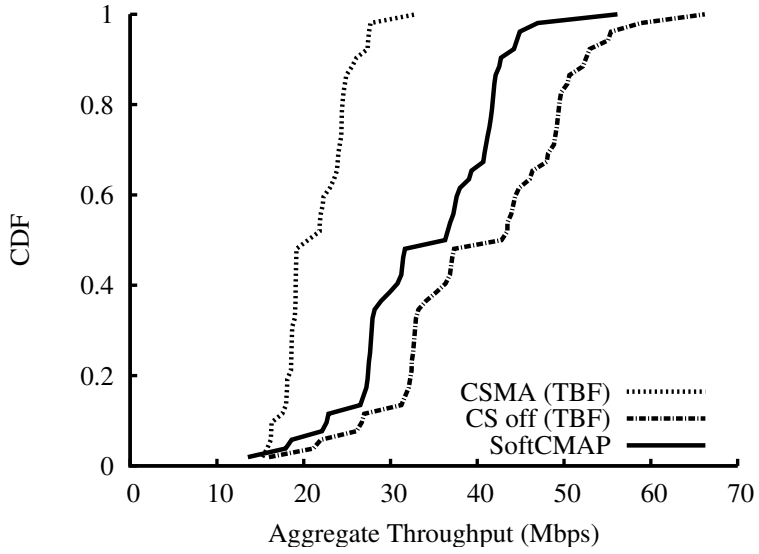


Figure 5-17: Aggregate throughput of two transmissions in trace-driven simulations with exposed terminals, comparing SoftCMAP, CS on, and CS off. The CS on and CS off 802.11 protocols are modified to implement time-based fairness (marked as TBF in the graph). The CDF is plotted over 50 experiments with different sets of nodes.

dian throughput improvement in this case was just $1.5\times$, significantly lower than CMAP with a window of eight virtual packets, because ACKs collided frequently at the senders and caused the senders to timeout and perform spurious retransmissions.

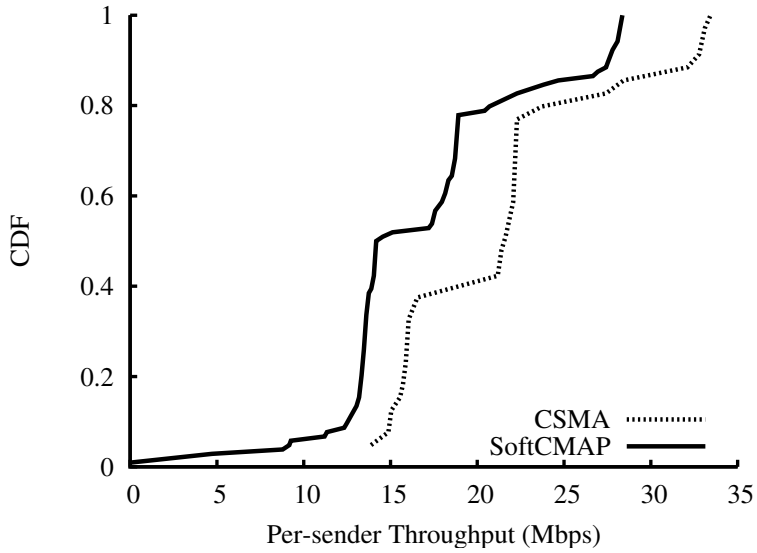


Figure 5-18: Per-sender throughput computed over the periods that the node is actively transmitting in trace-driven simulations of exposed terminals, comparing SoftCMAP and CSMA with time-based fairness. The CDF is plotted over 50 experiments with different sets of nodes.

Gains with SoftCMAP. We now investigate the impact of the conflict map mechanism on exposed terminals when each sender is capable of performing bit rate adaptation along with concurrent transmissions using the SoftCMAP protocol. We simulate the performance of 802.11 (with carrier sense turned on and off) and SoftCMAP on traces from experiments with pairs of exposed senders running at bit rates corresponding to the 802.11a rates of 6–36 Mbps. We run such experiments for 50 sets of nodes (16 unique sets of four nodes each, transmitting at 4 or 5 different transmit power levels). The exposed senders were chosen such that they were able to transmit concurrently with virtually no packet losses at the lowest bit rate of 6 Mbps, but may not be perfectly exposed in this manner at higher bit rates.

Figure 5-16 shows the CDF of the aggregate throughput of SoftCMAP, CSMA, and CS off over the 50 pairs of exposed senders. We see from the figure that the median improvement in throughput from SoftCMAP is close to $2\times$. However, some gains of SoftCMAP could simply be an artifact of SoftCMAP giving equal airtime to senders contending at different bit rates, resulting in senders at higher bit rates getting more airtime than with CSMA. We therefore compare SoftCMAP to flavors of 802.11 that implement time-based fairness, by letting contending senders repacketize data in frames with equal transmission times at all bit rates. We use the same transmission slot size in all protocols. Figure 5-17 compares the throughput of SoftCMAP against CSMA and CS off with this modification in place. We find that exploiting exposed terminals in SoftCMAP leads to a $1.7\times$ median improvement even over CSMA that implements time-based fairness, showing that these gains are purely from exploiting concurrency. We find that the performance of SoftCMAP is lower than that of carrier sense turned off because of necessary overheads such as waiting to decode the header before starting concurrent transmissions. In that sense, while the per-

formance of the CS off curve may not be achieved by any practical protocol, it serves as an upper bound on the possible improvement in performance and shows that the performance of SoftCMAP is not far away from optimal.

Figure 5-18 shows a CDF of the average throughput of the individual senders in the exposed terminal pair from the previous experiment, computed only over the periods that the node is actually transmitting. We see from the figure that this per-sender throughput is typically lower with SoftCMAP than with CSMA, because senders see a higher loss rate, or have to choose a lower bit rate, when transmitting with the added interference of a concurrent transmission in SoftCMAP. However, the average link throughput of the transmission is still higher with SoftCMAP than with CSMA because the exposed senders get many more transmission opportunities by transmitting concurrently with each other.

5.5.4 Performance Gains with Hidden Terminals

The conflict map mechanisms in CMAP will fail to detect a conflict when either (a) the receiver is unable to receive at least *some* headers and trailers from an interferer in order to populate its interferer list, and thereby the conflict map, or (b) a potential sender cannot hear the interferer's transmission headers in order to defer to it (the hidden terminal problem). As a result, an interferer that is out of communication range of either the receiver or the sender of a transmission (a "hidden interferer") can degrade the throughput of that transmission. This section investigates the impact of hidden interferers on the throughput of CMAP.

To identify the frequency of hidden interferers, we quantify the relationship between an interferer I's effect on the throughput of a transmission $S \rightarrow R$ and the probability that S and R can hear I. Our goal is to see how often it is that I effects the throughput of $S \rightarrow R$ but is not within hearing range of either. We randomly choose 500 802.11 sender-receiver pairs (S, R) with a potential transmission link between them, and for each pair, we pick an interferer I at random from all the nodes in the testbed. We first measure the throughput of the link $S \rightarrow R$ and the frame delivery rates of the links $I \rightarrow R$ and $I \rightarrow S$ in the absence of any other interference. S and I then transmit 802.11 packets continuously and simultaneously with carrier sense and link-layer ACKs disabled, and we measure the throughput of $S \rightarrow R$ in the presence of I's transmissions. We disable link-layer ACKs to avoid the senders backing off in response to packet losses. From the resulting data, we find that the fraction of cases where I reduces the throughput of $S \rightarrow R$ by more than 50%, but has a link with frame delivery rate less than 0.5 to either S or R is only 8%. This observation convinces us that hidden interferers do not occur very frequently in our experiments.

We now estimate the magnitude of impact a hidden interferer has on CMAP throughput as follows. Let p_r and p_s denote the packet reception rates of the links $I \rightarrow R$ and $I \rightarrow S$ respectively. Then the probability that both S and R hear I is at least $p = max(p_r + p_s - 1, 0)$. Let T denote the normalized throughput of $S \rightarrow R$ in the presence of I. Now, had the experiment been run with CMAP, the conflict detection mechanism would have avoided a throughput degradation (i.e., resulted in a normalized throughput of 1) with a probability p, and resulted in a lower throughput of T with a probability $1 - p$. Hence, the expected throughput of $S \rightarrow R$ with CMAP can be computed as $p \cdot 1 + (1 - p) \cdot T$. A sum of the

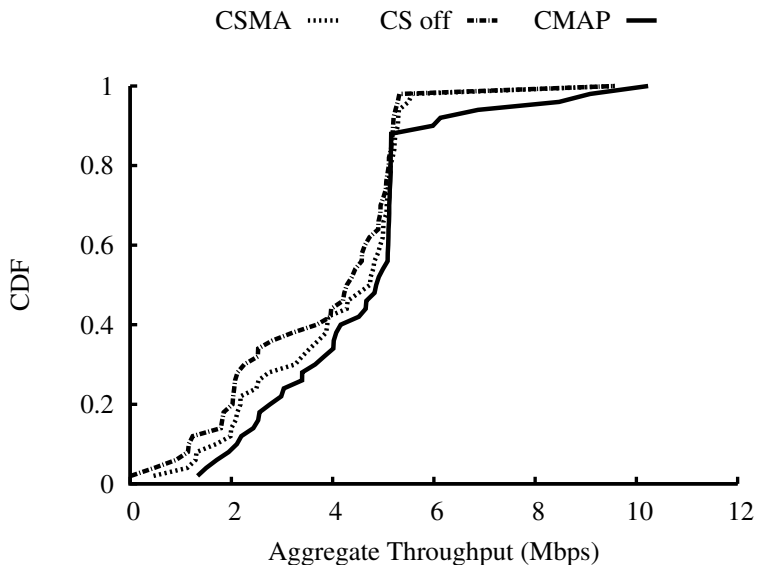


Figure 5-19: Aggregate throughput of two transmissions in an experiment with two senders out of each other's transmission range, comparing CMAP, CSMA, and CS off. The CDF is plotted over 50 experiments with different sets of nodes.

above expression over all data points works out to be 0.896. Thus, the expected damage to a CMAP pair's throughput due to a hidden interferer is only around 10%. In practice, however, the degradation will be even smaller (as we will see next) because CMAP senders back off in response to losses, unlike the senders in the above experiment which were made to transmit continuously.

We now evaluate how well CMAP's backoff protocol prevents performance degradation when the defer mechanism fails in an experiment with pairs of hidden terminals. We choose pairs of links for this experiment as shown in Figure 5-12(c): each receiver has a potential transmission link to both senders, ensuring that the two transmissions will almost always interfere with each other at the receivers. The senders are not in range of each other with the result that they cannot defer to each other's transmissions. Figure 5-19 presents the distribution of throughput across 50 randomly chosen link pairs. We see that CMAP and 802.11 (with both carrier sense enabled and disabled) perform comparably. Also note that there is very little weight on the right-hand side of the CDF that represents throughputs greater than the single pair throughput. This is because the best we can hope for in such topologies, with current 802.11 hardware, is transmissions interleaved with each other to achieve the throughput of a single sender-receiver pair. Therefore, while the conflict map mechanism does not explicitly solve the hidden terminal problem, it does not lead to a degradation of throughput compared to CSMA either.

We also use the above experiment to validate our design decision of transmitting both headers and trailers (as opposed to only headers) on packets. For each experiment in Figures 5-14 and 5-19, we compute the fraction of virtual packets transmitted by a sender for which either the header or the trailer was successfully received by the receiver. We compare this fraction against the fraction of virtual packets for which the header was received. We plot a CDF of these fractions across all sender-receiver pairs in each experiment in Fig-

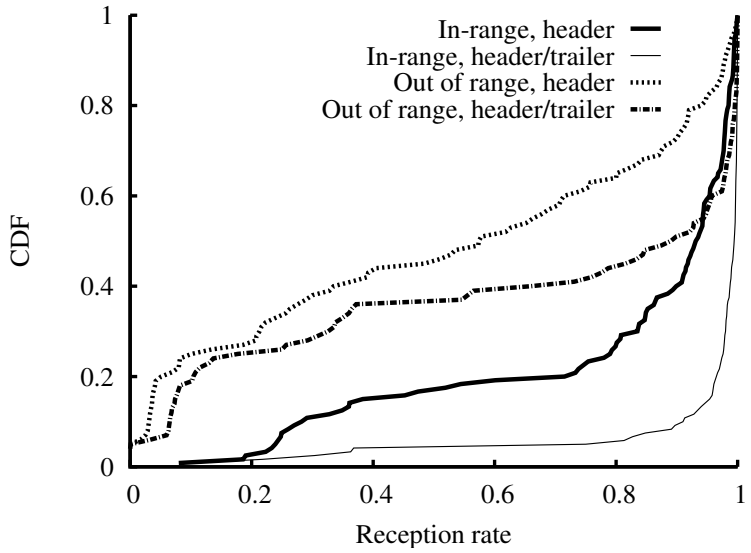


Figure 5-20: A CDF of the probabilities of reception of either header or trailer and header alone for each transmitted virtual packet, computed from the experiments with two pairs of senders in range (Figure 5-14) and out of range (Figure 5-19).

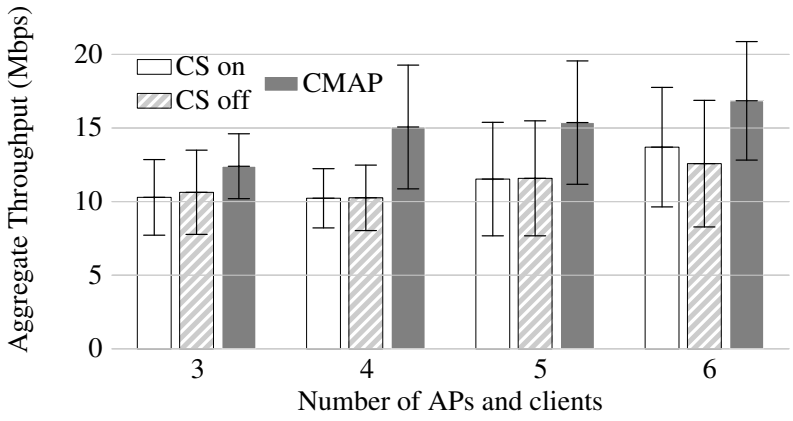


Figure 5-21: Mean throughput in the experiment with N APs and N clients, comparing CMAP, CSMA, and CS off.

ure 5-20. We see that the probability of reception of a header or trailer is higher than the probability of reception of a header alone in both the experiments; the benefit of using trailers is more pronounced when the senders were out of each other's range and persistently collided at the receivers. We also observe that the probability that either a header or trailer is received is almost 1 in the experiment where the senders are in range of each other and transmit equal-sized packets.

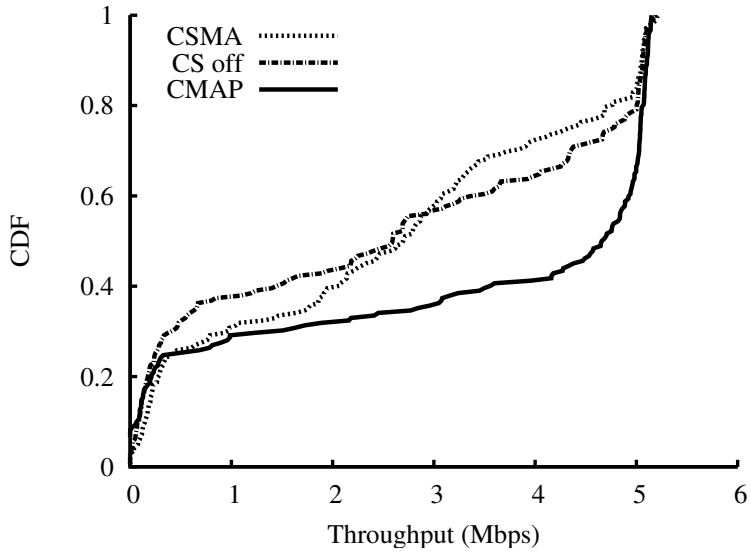


Figure 5-22: A CDF of the per-sender throughput in experiments with N APs and N clients, comparing CMAP, CSMA, and CS off.

5.5.5 Multi-node Topologies

In this section, we evaluate CMAP in topologies with more than two senders.

Access point network. We first pick topologies that resemble wireless local area networks (WLANs) with multiple access points (APs) and clients that span a geographical area several radio-ranges in diameter. We divide the testbed (see Figure 5-11) into six "regions" and designate one node in each region as an AP, such that each AP is out of the communication range of every other AP. We choose clients of an AP from the set of nodes in that region that have a potential transmission link to that AP, and randomly designate one of the client or AP as the sender of packets. We then perform experiments by varying the number of APs (N) from three through six, always choosing APs from adjacent regions when there are fewer than six APs in an experiment. Note that in each experiment, some of the N concurrent senders will be in hearing range of a receiver (thereby interfering directly with the receiver's reception) while some simply increase the perceived background noise. For each value of N, we perform 10 experiments, choosing different clients for APs each time.

Figure 5-21 shows the average aggregate throughput of CMAP and 802.11 (with both carrier sense enabled and disabled) as a function of the number of concurrent senders N in the experiment. Note that increasing the number of senders does not mean nodes perform more than two concurrent transmissions at a time. As we vary N, we find that CMAP improves aggregate throughput by between 21% (when $N = 3$) and 47% (when $N = 4$). CMAP sees this improvement because pairs of senders in adjacent cells were often exposed terminals. Figure 5-22 shows a CDF of the per-sender throughput for all senders across all experiments. From the figure we find that CMAP improves the median per-sender throughput by $1.8\times$ as compared to CSMA.

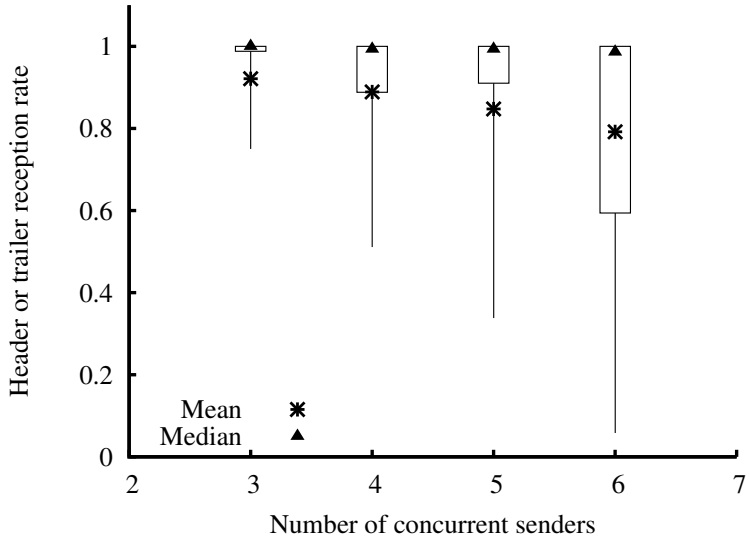


Figure 5-23: Mean and median probabilities of reception of either header or trailer of a virtual packet at a receiver, as a function of the number of concurrent senders in the experiment with N APs and clients. The boxed error bars represent the 25th and 75th percentiles, and the thin error bars show the 10th and 90th percentile values.

We next study how the header or trailer reception probabilities at a node are affected by the number of concurrent transmissions in the network. Figure 5-23 shows the mean, median, and various percentile values of the probability of reception of either a header or trailer of a virtual packet at each receiver, as a function of the number of concurrent transmissions in the network. We find from the graph that the median header or trailer reception probability is practically unaffected by the number of concurrent transmissions. However, the 10th percentile value drops sharply, indicating that a small fraction of receivers cannot implement the conflict map mechanisms effectively in the presence of many concurrent transmissions. This is because concurrent transmissions in CMAP are unaligned and often corrupt the headers and trailers of receptions. On the other hand, with the fine-grained timing control and the aligned concurrent transmissions of SoftCMAP, the concurrent transmissions do not overlap over the headers and trailers. This is one of the shortcomings of the CMAP protocol as compared to SoftCMAP.

Mesh network. In this section, we present an evaluation of CMAP over a two-hop content dissemination mesh network shown in Figure 5-12(d). The source S first broadcasts a batch of packets to its one-hop neighbors A_1, A_2, and A_3. The A_is then transmit the packets to the corresponding B_is. We compute the throughput at each B_i as the minimum of the throughputs along the corresponding $S \rightarrow A_i$ and $A_i \rightarrow B_i$ paths. We measured the aggregate throughput at all the B_is over 10 different topologies. We found that CMAP achieves a 52% higher average throughput than 802.11 with carrier sense enabled. The reason for this improvement was that, frequently, one or more of the A_is were exposed terminals during the $A_i \rightarrow B_i$ transfers. While the above experiment may not be representative of the

performance of CMAP over arbitrary mesh networks, it convinces us that multi-hop mesh networks can benefit from a MAC that exploits concurrent transmission opportunities. In fact, given MACs like SoftCMAP or CMAP that increase concurrency, routing protocols can be redesigned to generate topologies that increase concurrent transmission opportunities.

5.6 Related Work

There has been a lot of prior work on exploiting concurrency to improve performance of wireless networks. We classify the work into: (a) research that is directly related to ours, i.e., solves the exposed terminal problem using techniques different from ours, (b) research that is complementary to ours and can work synergetically with our approach, and (c) other work broadly related to ours.

The exposed terminal problem. Spatial reuse is a well-known concept in wireless communications networks of many different types. MACA [32] makes the observation that carrier sense cannot make correct transmission decisions because it does not consider channel conditions at the receiver, resulting in problems like exposed and hidden terminals. The paper proposes the RTS/CTS virtual carrier sensing protocol to solve the hidden terminal problem. However, this mechanism does not solve the exposed terminal problem.

There have been a few previous proposals to increase concurrency in wireless networks [4, 44, 58, 14]. As we explain below, however, our protocols differ from all these schemes in the method of identifying and exploiting the identified concurrent transmission opportunities. Also, none of these schemes considers heterogeneous bit rates or performs joint channel access and rate adaptation, a problem solved by SoftCMAP. Finally, these previous proposals build upon the rarely used RTS/CTS mechanism and evaluate their ideas in simulation alone. We will now describe these related schemes in more detail. Like the conflict map protocols, the "adaptive learning" extension of MACA-P [4] builds a data structure containing potentially non-interfering but nearby nodes. Unlike our work, however, MACA-P is based on the RTS/CTS exchange, extended in time to include a *control gap*, which results in a significant protocol overhead. RTSS/CTSS [44] uses an offline training phase to determine which nodes may transmit concurrently. This approach, however, is not applicable when the channel varies, as is the case in practice. It also does not have any mechanisms to deal with the ACK loss problem in exposed terminals. Shukla et al. [58] propose identifying exposed terminals performing an RTS/CTS exchange on the basis of overhearing an RTS without overhearing a CTS. This method does not identify all exposed terminal opportunities—it misses exposed terminals where a sender can hear another receiver's CTS, but is far enough from the receiver that it can transmit concurrently. In the Interference Aware (IA) MAC protocol [14], nodes make transmission decisions using the SINR estimates at receivers that are embedded in CTS messages. However, the IA MAC misses exposed terminals where one of the exposed senders does not hear the CTS from the other receiver.

Padhye et al. [48] propose a set of metrics that estimate link interference in static multihop wireless networks. They suggest an offline process of pairwise link measurements

to identify conflicting transmissions. Similarly, the interference map [46] builds up packet success information about CSMA transmissions in an offline training phase, for the purpose of network planning. Our work, in contrast, obtains interference information online and uses it in a live protocol.

Complementary techniques. Researchers have observed that concurrent transmissions do not always result in both the colliding packets being lost [60, 70]. This phenomenon, in which a receiver can correctly decode it's sender's packet even in the presence of other concurrent transmissions, is sometimes referred to as the "capture" effect. Our work increases the opportunities for and exploits packet capture by increasing the number of concurrent transmissions.

On older wireless hardware, a concurrent transmission may not be successful between exposed terminals if both the receivers synchronize with the transmission that starts first and ignore the preamble of the subsequent transmission. Whitehouse et al. [70] and Priyantha [49] propose mechanisms to boost the chances of packet capture in such cases by making receivers acquire packet preambles throughout the duration of ongoing packet receptions in addition to when no packet is being received. That is, receivers can capture stronger transmissions that start during the reception of weaker transmissions. With this capability of the PHY (that is common in newer hardware), other researchers have proposed that one can order concurrent transmissions carefully to boost the probability of successfully decoding both the transmissions at the corresponding receivers [42]. Deploying such a mechanism with conflict maps increases the number of exposed terminal opportunities available to exploit. Our CMAP prototype built on commodity wireless hardware does not resynchronize during a reception, while our software radio prototype used to evaluate SoftCMAP was programmed to do so.

Successive interference cancellation [22, 19] performs concurrent transmissions that interfere with each other, but can be jointly decoded using physical layer techniques. Techniques that use multiple antennas in MIMO systems to decode concurrent transmissions [64] also use physical layer techniques to decode transmissions that would normally interfere. The conflict map mechanism can be used along with such techniques to help them schedule concurrent transmissions that can be successfully decoded at the physical layer. That is, the streaming PHY abstraction can be used to perform concurrent transmissions not only in the case of exposed terminals, but also in the case of other types of concurrent transmissions that can succeed in parallel.

Other related work. In a busy access point WiFi network, Judd [30] observes that many clients connected to different access points are in fact exposed terminals with respect to each other. In fact, 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 result suggests that the use of conflict maps could significantly improve performance in infrastructure wireless LANs.

Recent research [12] has claimed that the added benefits of exploiting exposed terminals when using a smart rate adaptation protocol are negligible in practice because senders are often not exposed terminals when transmitting close to their channel capacity. Our

experiments with SoftCMAP, however, convince us that smart rate adaptation does not preclude the possibility of throughput gains from exploiting exposed terminals. The difference in conclusions can possibly be explained by the different hardware the experiments were run on. Our software radio receivers used to perform concurrent transmissions in the evaluation of SoftCMAP are capable of resynchronizing to a stronger preamble while decoding a transmission. Older commodity hardware generally does not have this capability, as a result of which the incidence and impact of the exposed terminal problem may appear lower.

Algorithms to tune the carrier sense threshold or power level alone [73, 76, 45, 59, 69, 6, 62] and algorithms that tune both carrier sense threshold and transmit power [35] build on the basic carrier sense mechanism. Thus, these algorithms make a fundamental tradeoff between preventing hidden-terminal collisions and permitting exposed-terminal spatial reuse, and don't fully take advantage of the many exposed terminal opportunities present in real networks. Our work explicitly discriminates between conflicting and non-conflicting transmissions, avoiding this tradeoff.

There have also been proposals to use receiver-based feedback of channel conditions in making transmission decisions to improve the performance of CSMA. E-CSMA [17] 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. The distinguishing feature of our work from E-CSMA is that we explicitly takes the identity of current senders and whom they're sending to into account while making channel access decisions, instead of implicitly capturing them using signal strength estimates, and hence can better predict which transmissions are likely to succeed and which not.

5.7 Chapter Summary

This chapter described two channel access protocols, SoftCMAP and CMAP, that aim to improve MAC-layer throughput by increasing the number of successful concurrent transmissions. The protocols are based on the idea of conflict maps, where nodes base their channel access decisions on empirical measurements of which transmissions conflict and which do not, and exploit all possible concurrent transmission opportunities. The key enhancement to the MAC-PHY interface that permits the implementation of these protocols is the notion of a streaming PHY, where nodes can learn of the identities of the senders and receivers of ongoing transmissions and make channel access decisions based on who is transmitting and not on whether anyone is transmitting. The empirical measurements used to build the conflict map are frame loss rates in the case of CMAP and channel BERs from SoftPHY hints in the case of SoftCMAP. SoftCMAP also performs a joint channel access and bit rate decision using a rate adaptation protocol similar to SoftRate—a sender picks the transmit bit rate based on whether there is another concurrent transmission or not. Our evaluation of the two protocols shows that one can obtain close to 50% increase in aggregate network throughput by exploiting concurrent transmission opportunities in wireless networks.

Chapter 6

Conclusion

We conclude the dissertation with a summary of our contributions and a discussion of directions for future research.

6.1 Summary

We presented a cross-layer approach to link-layer protocol design in wireless networks. The wireless channel is a time-varying broadcast medium, and the wireless link layer abstracts most of this complexity from higher layers. In today's network architecture, link-layer protocols glean only a small amount of information about channel quality (like frame receptions and per-frame SNR measurements) from the PHY. We showed that this information sometimes turns out to be insufficient in capturing the channel correctly, leading to incorrect transmit bit rate and channel access decisions.

This dissertation proposed a new interface between the physical and link layers in the wireless networking stack to export more information about the wireless channel to the link layer. With our interface, the PHY streams decoded bits along with per-bit confidences called SoftPHY hints to the link layer. The SoftPHY hints of the bits in a frame can be used to accurately compute the underlying wireless channel BER, a fundamental metric of interest for bit rate adaptation protocols. The streaming interface enables the link layer to learn about ongoing transmissions in a timely manner and make appropriate channel access and transmit bit rate decisions. Our design of the new interface works across a wide class of physical layer architectures.

We also described new bit rate adaptation and MAC protocols that use the information available through this interface to improve performance. The SoftRate bit rate adaptation protocol uses the channel BER computed from SoftPHY hints as the feedback signal to pick suitable transmit bit rates. SoftRate also uses patterns of SoftPHY hints to identify and eliminate the effects of transient interference on its channel quality estimate. SoftRate improves application-layer throughput by 35% to 2$\times$ over existing rate adaptation protocols because of its ability to estimate the wireless channel quickly and accurately across many different operating environments. SoftRate is particularly useful in fast-varying channels (e.g., when the sender or receiver or both are in a moving vehicle) or channels that experience interference losses (e.g., crowded conference rooms).

With the SoftCMAP MAC protocol, senders use the streaming PHY to decode headers of ongoing transmissions and transmit concurrently with exposed terminals, increasing the number of successful concurrent transmissions. Senders identify other exposed senders using the conflict map, a distributed map of conflicting transmissions constructed using feedback from receivers. Senders also use a rate adaptation protocol like SoftRate to pick suitable transmit bit rates for concurrent transmissions. In other words, the SoftCMAP protocol allows senders to make a jointly-optimal transmit bit rate and channel access decision. We also developed the CMAP protocol, an approximation of SoftCMAP that runs on commodity hardware. These protocols based on conflict maps increase spatial reuse and improve aggregate throughput by up to 50% in typical networks.

6.2 Future Work

Real-world deployment. The dissertation opens up many avenues for future research. Foremost among them is validating the ideas presented here in bigger networks and real-world deployments. The experiments in this dissertation, as is the case with most cross-layer protocol evaluations, were predominantly carried out with software radios that work at link-speeds that are lesser than those seen in real networks. An interesting research question is to verify whether these gains hold at link speeds of many hundreds of megabits per second. Such an evaluation requires an end-to-end implementation of the link-layer protocols in hardware, using a high-speed wireless networking platform running on FPGAs or ASICs. A hardware implementation with a compact form factor will also enable testing the protocols in large deployments with real traffic, possibly using vehicular testbeds [18] that encounter a highly time-varying wireless channel.

Other uses of SoftPHY hints. SoftPHY hints estimate the quality of a frame reception, and can be quite useful to higher-layer functions besides bit rate adaptation. One possible application is in Hybrid ARQ (HARQ) error recovery schemes. In HARQ with incremental redundancy, the sender first transmits only a subset of the parity bits in a coded bit stream, and transmits the other parity bits only if the initial transmission fails. Receivers jointly decode all the transmissions of a frame, and request retransmissions till the frame is correctly decoded. Current HARQ schemes have no easy way to estimate how bad the initial transmission was and how many parity bits the receiver needs to be able to decode it correctly. With the SoftPHY interface, the receiver's link layer can use SoftPHY hints to estimate how many bits are likely to be in error and request an appropriate size of retransmission of parity bits to enable correct decoding. SoftPHY hints can also identify specific portions of the packet with low confidences, and the receiver can possibly request different number of parity bits over different portions of the packet.

Other uses of the streaming PHY. Recall that in the exposed terminal problem, the PHY can decode a transmission in the presence of another concurrent transmission because the concurrent transmission adds only a negligible amount of interference at the receivers of the exposed senders. However, by using advanced signal processing techniques, some modern PHYs today can decode concurrent transmissions even when both the transmissions

interfere with each other. For example, with Successive Interference Cancellation (SIC), a receiver first decodes the stronger of the two transmissions in a pair of concurrent transmissions, subtracts the effect of the decoded transmission from the signal, and then decodes the second transmission. Another example is multi-user MIMO, where a receiver (say, an access point) uses multiple antennas to communicate with multiple senders simultaneously. Using joint decoding algorithms, a receiver can decode as many concurrent data streams as the number of antennas it has. All these PHY techniques require a MAC protocol that enables the senders to transmit concurrently and reap the benefits of the advanced PHY at the receiver. But because these techniques have been developed in the context of cellular communication systems, the only MAC protocols that work with such PHYs today are centralized protocols with a controller scheduling concurrent transmissions. In order to apply these PHY techniques in the context of a data network with bursty traffic, one needs to come up with a distributed contention-based MAC protocol that enables senders to transmit concurrently with selected senders. The ideas in SoftCMAP can be used to address this problem. For example, a conflict is redefined as a pair of concurrent transmissions where SIC will not work at either receiver. Nodes then use the streaming PHY and the conflict map to transmit concurrently with transmissions for which SIC is likely to work at the receivers, picking appropriate bit rates based on the channel access decision. One can see that the streaming PHY has applications beyond just the exposed terminal problem, and can be used to develop link-layer protocols that treat concurrent transmissions as first-class citizens.

In summary, this dissertation demonstrated the power of cross-layer techniques in the design of wireless bit rate adaptation and channel access protocols. By exposing fine-grained information about the dynamic wireless channel via well-defined interfaces, one can greatly improve throughput in wireless networks. Looking ahead, the philosophy of this dissertation can be extended to develop cross-layer solutions to other open problems in the link and higher layers in wireless networks.

Bibliography

[1] Bluespec, Inc. `http://www.bluespec.com`.

[2] The GNU Radio Software Radio. `http://gnuradio.org/trac`.

[3] The ns-3 Network Simulator. `http://www.nsnam.org`.

[4] A. Acharya, A. Misra, and S. Bansal. Design and Analysis of a Cooperative Medium Access Scheme for Wireless Mesh Networks. In *Proceedings of IEEE BROADNETS*, pages 621–631, San Jose, CA, October 2004.

[5] P. A. K. Acharya, A. Sharma, E. M. Belding, K. C. Almeroth, and D. Papagiannaki. Congestion-Aware Rate Adaptation in Wireless Networks: A Measurement-Driven Approach. In *Proceedings of IEEE SECON Conference*, pages 1–9, San Francisco, CA, June 2008.

[6] Sharad Agarwal, Srikanth V. Krishnamurthy, Randy H. Katz, and Son K. Dao. Distributed Power Control in Ad-hoc Wireless Networks. In *Proceedings of PIMRC*, pages 59–66, San Diego, CA, September 2001.

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

[8] C. Berrou and A. Glavieux. Near optimum error correcting coding and decoding: Turbo-codes. *IEEE Transactions on Communications*, 44:1261–1271, October 1996.

[9] Pravin Bhagwat, Partha Bhattacharya, Arvind Krishna, and Satish Tripathi. Using Channel State Dependent Packet Scheduling to Improve TCP Throughput over Wireless LANs. *Wireless Networks*, 1997.

[10] S. Biaz and N. H. Vaidya. Discriminating Congestion Losses from Wireless Losses Using Inter-arrival Times at the Receiver. In *Proceedings of the IEEE ASSET Symposium*, pages 10–17, Richardson, TX, March 1999.

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

[12] Micah Z. Brodsky and Robert T. Morris. In Defense of Wireless Carrier Sense. In *Proceedings of ACM SIGCOMM*, Barcelona, Spain, 2009.

[13] J. Camp and E. Knightly. Modulation Rate Adaptation in Urban and Vehicular Environments: Cross-Layer Implementation and Experimental Evaluation. In *Proceedings of the ACM MobiCom*, pages 315–326, San Francisco, CA, September 2008.

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

[15] Ranveer Chandra, Ratul Mahajan, Thomas Moscibroda, Ramya Raghavendra, and Paramvir Bahl. A Case for Adapting Channel Width in Wireless Networks. In *Proceedings of ACM SIGCOMM*, Seattle, WA, August 2008.

[16] F. Edalat. *Real-time Sub-carrier Adaptive Modulation and Coding in Wideband Orthogonal Frequency Division Multiplexing Wireless Systems*. PhD thesis, Massachusetts Institute of Technology, December 2007. `http://hdl.handle.net/1721.1/43031`.

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

[18] Jakob Eriksson, Hari Balakrishnan, and Samuel Madden. Cabernet: Vehicular Content Delivery Using WiFi. In *Proceedings of ACM MobiCom*, San Francisco, CA, September 2008.

[19] S. Gollakota and D. Katabi. Zigzag Decoding: Combating Hidden Terminals in Wireless Networks. In *Proceedings of the ACM SIGCOMM*, pages 159–170, Seattle, WA, August 2008.

[20] Ramakrishna Gummadi, Rabin Patra, Hari Balakrishnan, and Eric Brewer. Interference Avoidance and Control. In *Proceedings of Hotnets-VII*, Calgary, Canada, October 2008.

[21] Joachim Hagenauer and Peter Hoeher. A Viterbi Algorithm with Soft-Decision Outputs and its Applications. In *Proceedings of IEEE GLOBECOM*, pages 1680–1686, Dallas, TX, November 1989.

[22] D. Halperin, T. Anderson, and D. Wetherall. Taking the Sting out of Carrier Sense: Interference Canncelation for Wireless LANs. In *Proceedings of ACM MobiCom*, pages 339–350, San Francisco, CA, September 2008.

[23] G. Holland, N. Vaidya, and P. Bahl. A Rate-Adaptive MAC Protocol for Multihop Wireless Networks. In *Proceedings of ACM MobiCom Conference*, pages 236–251, Rome, Italy, September 2001.

[24] Hyunsoo Cheon and Daesik Hong. Effect of Channel Estimation Error in OFDM-Based WLAN. *IEEE Communications Letters*, May 2002.

[25] IEEE Standard 802.16e-2005: Air Interface for Fixed and Mobile Broadband Wireless Access Systems, Amendment 2, February 2006. http://standards.ieee.org/getieee802/802.16.html.

[26] Kyle Jamieson. *The SoftPHY Abstraction: from Packets to Symbols in Wireless Network Design*. PhD thesis, Massachusetts Institute of Technology, June 2008. http://hdl.handle.net/1721.1/41857.

[27] Kyle Jamieson and Hari Balakrishnan. PPR: Partial Packet Recovery for Wireless Networks. In *Proceedings of ACM SIGCOMM*, pages 409–420, Kyoto, Japan, August 2007.

[28] G. D. Forney Jr. The Viterbi Algorithm. In *Proceedings of the IEEE*, volume 61, pages 268–278. IEEE, March 1973.

[29] G. Judd, X. Wang, and P. Steenkiste. Efficient Channel-aware Rate Adaptation in Dynamic Environments. In *Proceedings of ACM MobiSys*, pages 118–131, Breckenridge, CO, June 2008.

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

[31] 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.

[32] P. Karn. MACA - A New Channel Access Method for Packet Radio. In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, September 1990.

[33] Sachin Katti, Dina Katabi, Hari Balakrishnan, and Muriel Medard. Symbol-Level Network Coding for Wireless Mesh Networks. In *Proceedings of ACM SIGCOMM*, Seattle, WA, August 2008.

[34] J. Kim, S. Kim, S. Choi, and D. Qiao. CARA: Collision-Aware Rate Adaptation for IEEE 802.11 WLANs. In *Proceedings of IEEE INFOCOM*, Barcelona, Spain, April 2006.

[35] Tae-Suk Kim, Hyuk Lim, and Jennifer Hou. Improving Spatial Reuse through Tuning Transmit Power Carrier Sense Threshold and Data Rate in Multihop Wireless Networks. In *Proceedings of ACM MobiCom*, pages 366–377, Los Angeles, CA, September 2006.

[36] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[37] M. Lacage, M. H. Manshaei, and T. Turletti. IEEE 802.11 Rate Adaptation: A Practical Approach. In *Proceedings of ACM MSWiM Workshop*, pages 126–134, Venezia, Italy, October 2004.

[38] K. C. Lin, N. Kushman, and D. Katabi. ZipTx: Exploiting the Gap Between Bit Errors and Packet Loss. In *Proceedings of ACM MobiCom*, pages 351–362, San Francisco, CA, September 2008.

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

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

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

[42] Justin Manweiler, Naveen Santhapuri, Souvik Sen, Romit Roy Choudhury, Srihari Nelakuditi, and Kamesh Munagala. Order Matters: Transmission Reordering in Wireless Networks. In *Proceedings of ACM MobiCom*, Beijing, China, September 2009.

[43] J. Metzner. Improvements in Block-Retransmission Schemes. *IEEE Transactions on Communications*, 27(2):524–532, February 1979.

[44] K. Mittal and E. Belding. RTSS/CTSS: Mitigation of Exposed Terminals in Static 802.11-Based Mesh Networks. In *Proceedings of IEEE WiMesh Workshop*, Reston, VA, September 2006.

[45] J. Monks, V. Bharghavan, and W. Hwu. A Power Controlled Multiple Access Protocol for Wireless Packet Networks. In *Proceedings of IEEE INFOCOM*, pages 219–228, Anchorage, AK, April 2001.

[46] Dragoş Niculescu. Interference Map for 802.11 Networks. In *Proceedings of the ACM/USENIX Internet Measurement Conference*, San Diego, CA, October 2007.

[47] ONOE Rate Control. `http://madwifi.org/browser/trunk/ath_rate/onoe`.

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

[49] Nissanka Bodhi Priyantha. *The Cricket Indoor Location System*. PhD thesis, MIT, June 2005.

[50] John G. Proakis. *Digital Communications, 4th ed.* McGraw-Hill, 2000.

[51] Hariharan Rahul, Farinaz Edalat, Dina Katabi, and Charles Sodini. Frequency-Aware Rate Adaptation and MAC Protocols. In *Proceedings of ACM MobiCom*, Beijing, China, September 2009.

[52] Shravan Rayanchu, Arunesh Mishra, Dheeraj Agrawal, Sharad Saha, and Suman Banerjee. Diagnosing Wireless Packet Losses in 802.11: Separating Collision from Weak Signal. In *Proceedings of IEEE INFOCOM*, pages 735–743, Phoenix, AZ, April 2008.

[53] Patrick Robertson, Emmanuelle Villebrun, and Peter Hoeher. A Comparison of Optimal and Sub-optimal MAP Decoding Algorithms Operating in the Log Domain. In *Proceedings of IEEE ICC*, Seattle, WA, June 1995.

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

[55] Sanjiv Nanda and Kiran Rege. Frame Error Rates For Convolutional Codes On Fading Channels and the Concept of Effective Eb/No. *IEEE Transactions on Vehicular Technology*, 47:1245–1250, November 1998.

[56] T. M. Schmidl and D. C. Cox. Robust Frequency and Timing Synchroniation for OFDM. *IEEE Transactions on Communications*, 45:1613–1621, December 1997.

[57] Souvik Sen, Naveen Santhapuri, Romit Roy Choudhury, and Srihari Nelakuditi. CBAR: Constellation Based Rate Adaptation in Wireless Networks. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, April 2010.

[58] D. Shukla, L. Chandran-Wadia, and S. Iyer. Mitigating the Exposed Node Problem in IEEE 802.11 Ad Hoc Networks. In *Proceedings of IEEE ICCCN*, pages 157–162, Dallas, TX, October 2003.

[59] Dongjin Son, Bhaskar Krishnamachari, and John Heidemann. Experimental Study of the Effects of Transmission Power Control and Blacklisting in Wireless Sensor Networks. In *Proceedings of the IEEE Conference on Sensor and Ad-hoc Communication and Networks*, pages 289–298, Santa Clara, CA, October 2004.

[60] Dongjin Son, Bhaskar Krishnamachari, and John Heidemann. Experimental Analysis of Concurrent Packet Transmissions in Low-Power Wireless Networks. In *Proceedings of ACM SenSys*, pages 237–250, Boulder, CO, November 2006.

[61] Hui Song, Raymond Kwan, and Jie Zhang. On Statistical Characterization of EESM Effective SNR over Frequency Selective Channels. *IEEE Transactions on Wireless Communications*, August 2009.

[62] T. Albatt and A. Ephremides . Joint Scheduling and Power Control for Wireless Adhoc Networks. In *Proceedings of IEEE INFOCOM*, 2002.

[63] Godfrey Tan and John Guttag. Time-based Fairness Improves Performance in Multirate Wireless LANs. In *Proceedings of USENIX Annual Technical Conference*, Boston, MA, June 2004.

[64] Kun Tan, He Liu, Ji Fang, Wei Wang, Jiansong Zhang, Mi Chen, and Geoffrey M. Voelker. SAM: Enabling Practical Spatial Multiple Access in Wireless LAN. In *Proceedings of ACM MobiCom*, Beijing, China, September 2009.

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

[66] Andrew J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. In *IEEE Transactions on Information Theory*, 1967.

[67] Mythili Vutukuru, Hari Balakrishnan, and Kyle Jamieson. Cross-Layer Wireless Bit Rate Adaptation. In *Proceedings of ACM SIGCOMM*, Barcelona, Spain, August 2009.

[68] Mythili Vutukuru, Kyle Jamieson, and Hari Balakrishnan. Harnessing Exposed Terminals in Wireless Networks. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, April 2008.

[69] R. Wattenhofer, L. Li, P. Bahl, and Y. Wang. Distributed Toplogy Control for Power Efficient Operation in Multihop Wireless Ad Hoc Networks. In *Proceedings of IEEE INFOCOM*, pages 1388–1397, Anchorage, AK, April 2001.

[70] Kamin Whitehouse, Alec Woo, Fred Jiang, Joseph Polastre, and David Culler. Exploiting the Capture Effect for Collision Detection and Recovery. In *IEEE EmNets Workshop*, Sydney, Australia, May 2005.

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

[72] Grace R. Woo, Pouya Kheradpour, Dawei Shen, and Dina Katabi. Beyond the bits: Cooperative packet recovery using physical layer information. In *Proceedings of ACM MobiCom*, pages 147–158, Montreal, Quebec, Canada, September 2007.

[73] X. Yang and N. Vaidya. On Physical Carrier Sensing in Wireless Ad Hoc Networks. In *Proceedings of IEEE INFOCOM*, pages 2525–2535, Miami, FL, March 2005.

[74] J. Zhang, K. Tan, J. Zhao, H. Wu, and Y. Zhang. A Practical SNR-Guided Rate Adaptation. In *Proceedings of IEEE INFOCOM*, pages 2083–2091, Phoenix, AZ, April 2008.

[75] Yahong Zheng and Chengshan Xiao. Simulation Models With Correct Statistical Properties for Rayleigh Fading Channels. *IEEE Transactions on Communications*, 51(6):920–928, 2003.

[76] Jing Zhu, Xingang Guo, L. Lily Yang, W. Steven Conner, Sumit Roy, and Mousumi 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.