Hardware Implementation of Wireless Bit Rate Adaptation

by

Samuel A. Gross

S.B, Massachusetts Institute of Technology (2010)

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

Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author .....
Department of Electrical Engineering and Computer Science
September 13, 2010

Certified by .....
Hari Balakrishnan
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Dr. Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

Hardware Implementation of Wireless Bit Rate Adaptation

by

Samuel A. Gross

Submitted to the Department of Electrical Engineering and Computer Science
on September 13, 2010, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis presents a hardware implementation of the SoftRate bit-rate adaptation protocol. SoftRate is a new bit-rate adaptation protocol, which uses per-bit confidence hints generated by the convolutional decoder to estimate the channel bit-error rate.

Implementing SoftRate requires changes to both the physical and media access control layers, which precludes using existing commodity 802.11 hardware. This project developed a SoftRate implementation on top of Airblue, an FPGA platform for developing wireless protocols. We present a hardware implementation of SoftRate which meets 802.11 timing requirements.

Thesis Supervisor: Hari Balakrishnan
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to thank my advisor, Hari Balakrishnan, for his advice and guidance. I also thank Alfred Man Cheuk Ng, Kermin Elliott Fleming, and Mythili Vutukuru for their help with this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many wireless networks, including 802.11, can operate at multiple bit-rates. Higher bit-rates allow for higher data rates on high quality channels, but result in more packet losses on noisy channels. Bit-rate adaptation protocols attempt to select the bit-rate that maximizes overall throughput depending on channel quality estimates. The performance of protocols that select too high a bit-rate will suffer due to frame losses, while protocols that select too low a bit-rate will waste available bandwidth. Accurate bit-rate selection is important to fully utilizing modern wireless local area networks.

SoftRate is a new bit-rate adaptation protocol that uses per-bit confidence hints to estimate channel bit-error rate at the receiver. Compared with earlier bit-rate adaptation protocols, SoftRate is able to adapt more quickly to changing channel conditions and is able to better estimate conditions in fading channels [12].

Previous work [12] evaluated a software implementation of SoftRate offline using recorded traces and simulation. Evaluating SoftRate in a real-world environment will require hardware support due to the strict latency requirements of wireless networks. For SoftRate to be useful, it must be implemented efficiently on hardware.

The aim of this project is to develop and evaluate a hardware implementation of the SoftRate protocol that is able to meet the performance constraints of wireless local area networks. The hardware implementation is developed on Airblue, an FPGA-

based platform for developing wireless protocols. Airblue is described in detail in [7].

A hardware implementation of SoftRate is necessarily an approximation of the software implementation, due to the need to reduce hardware complexity and processing latency. The hardware implementation therefore serves two goals. First, it demonstrates that implementing SoftRate is feasible in hardware, with the approximations that such an implementation requires. Second, it serves as a stepping-stone towards a real-world evaluation of the SoftRate protocol.

This project contributes an FPGA-based implementation of the SoftRate protocol. We implemented a SoftRate medium access control (MAC) and tested two methods of efficiently computing the average bit-error rate in hardware. To evaluate the SoftRate implementation, we implemented an signal-to-noise ratio (SNR)-based rate adaptation protocol and a software channel simulator. We also developed an improved synchronizer for Airblue, which detects transmissions at low SNRs. In hardware simulation, the SoftRate implementation achieved achieved 10%-40% higher throughput than the SNR-based protocol across varying channel conditions. The SoftRate implementation is able to meet 802.11a timing specifications, including embedding feedback and transmitting an ACK within 25 μs of receiving a packet.

1.1 Hardware Considerations

Design of hardware systems requires different trade-offs than design of software systems. Algorithms that are efficient in software are not necessarily efficient in hardware. Hardware systems are able to exploit a much finer grained parallelism than software systems. For example, the modules in the Airblue pipeline are able to operate in parallel because each module has its own dedicated hardware resources. Designing for this sort of task-level parallelism requires accessing data in a streaming manner.

Some operations that are relatively inexpensive in software would be impractical in our system. For example, Airblue uses fixed-point rather than floating-point rep-

resentation to avoid the need for hardware intensive floating-point support at every stage in the pipeline.

1.2 Roadmap

The rest of this work is organized as follows. First, some background about wireless networks and bit-rate adaptation protocols is presented. Chapter 3 describes Airblue, the FPGA-based platform for developing wireless protocols used in this work. Chapter 4 describes the hardware implementation of the SoftRate protocol on the Airblue platform. Chapter 5 presents an evaluation of this implementation using a simulated channel model. Chapter 6 concludes this work.

Chapter 2

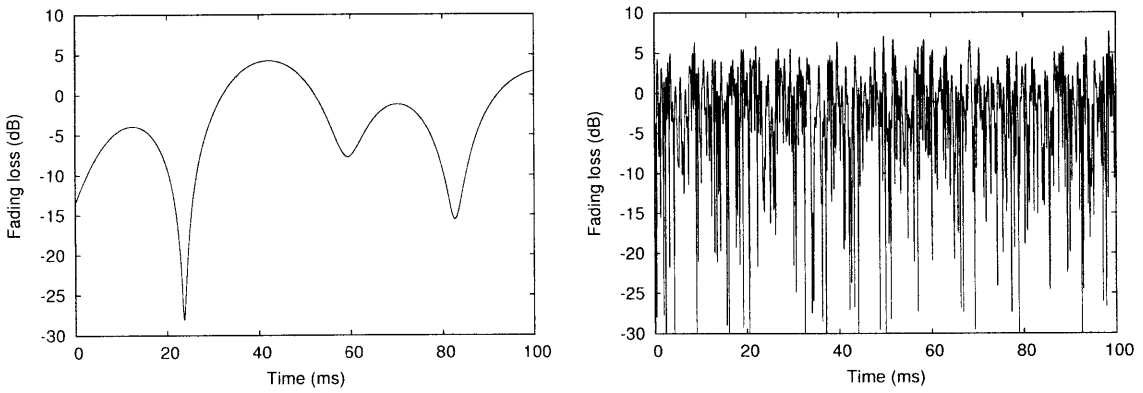
Background

Wireless networks suffer a variety of effects that degrade signal quality. Four effects are listed in [13]: interference, attenuation, shadowing, and fading. Interference occurs when multiple nodes transmit at the same time, resulting in corrupted signals at the receiver. Attenuation describes the power loss of a signal as it propagates through the environment. Power loss due to obstacles in the environment is called shadowing. Multipath fading occurs when multiple copies of the signal reach the receiver through different paths. These copies arrive with different phases and frequencies and can interfere constructively or destructively.

Fading effects vary with time as the sender, receiver, or objects in the environment move. The coherence time of a fading channel refers to the duration in which signals experience correlated effects. It is related to the carrier frequency and the speed of the sender, receiver, and objects in the environment. In the 5 GHz band used by 802.11a, walking speed (3 mph) results in a coherence time of about 20 ms. For comparison, 802.11a packet transmission typically lasts up to a few milliseconds. This type of channel, with a coherence time lasting multiple packet durations, is referred to as *slow fading*. Figure 2-1(a) shows the power loss in a simulated fading channel with a coherence time of 20 ms over a 100 ms interval.

Even when the sender and receiver are static, moving objects in the environment can cause fading. In [4], the authors found that passing cars can drive the coherence time of a channel down to as low as 300 μs in an urban environment. In this scenario,

channels effects vary even within a packet duration. These types of channels are referred to as *fast fading*. Figure 2-1(b) shows power loss in a simulated fading channel with a coherence time of 300 μs.



(a) Coherence time of 20 ms

(b) Coherence time of 300 μs

Figure 2-1: Fading in channels with coherence times of 20 ms and 300 μs over a 100 ms interval.

2.1 Bit-Rate Adaptation

The coherence time of fading channels has implications for bit-rate adaptation protocols. Depending on the coherence time, the protocol may need to pick a new rate every few packets. Frame-based rate adaptation protocols, including SampleRate [3], choose bit-rates based on the measured packet-loss rate aggregated over multiple packets. The performance of these protocols suffers when the channel coherence time lasts only the duration of a few packets [13].

SNR-based protocols use estimates of the signal-to-noise ratio (SNR) to choose transmit rate. For example, RBAR [5] uses an RTS/CTS exchange at the start of each packet to estimate the SNR. Fast fading channels cause problems for protocols like RBAR because channel conditions vary even within a single packet duration. Cellular physical layers (PHYs) often measure SNR continuously using pilot subcarriers to compute an average SNR over the entire frame. This method is more accurate in

measuring fast fading channels, but incurs the additional overhead of transmitting pilot symbols [13].

SoftRate uses confidence hints generated during convolutional decoding to estimate channel bit-error rate. The bit-error rate estimate is communicated to the sender in the time allotted for an 802.11 ACK, avoiding the need for a costly RTS/CTS exchange as in RBAR. Since SoftRate computes the bit-error rate (BER) estimate for individual frames, it is responsive to rapidly changing channel conditions [13].

However, unlike some previously proposed rate adaptation mechanisms, SoftRate requires modifications to both the 802.11 physical and medium access control layers. The next section gives an overview of these layers.

2.2 802.11

The IEEE 802.11 standard describes both the physical layer (PHY) and medium access control (MAC) layer. The MAC layer is responsible for sharing the wireless medium with other nodes. It decides when to transmit and at which rate to transmit packets. The SoftRate rate selection algorithm is implemented at the MAC layer.

The physical layer is responsible for the actual encoding, modulation, and transmission of packets across the wireless medium. SoftRate alters the standard decoding algorithm to generate SoftPHY confidence hints.

2.2.1 MAC Layer

The 802.11 MAC uses carrier sense multiple access with collision avoidance (CSMA/CA) to prevent nodes from transmitting at the same time. The MAC uses carrier sense to detect other transmissions and wait until they complete before transmitting. Exponential backoffs, a form of collision avoidance, are used to reduce the probability of repeated collisions. Wireless nodes should not reduce bit-rate in response to collisions because it interferes with these techniques and increases the chance of subsequent collisions due to longer packet transmission times.

The MAC uses a checksum at the end of each packet to verify that the packet was received without errors. SoftRate modifies data packets so that they also include a checksum after the header, which allows receivers to send feedback for packets with bit-errors in the body.

2.2.2 Physical Layer

The 802.11a/g specification uses orthogonal frequency-division multiplexing (OFDM) at the physical layer to modulate data. The 20 MHz wide channel is divided into 64 orthogonal sub-carriers, each 0.3125 MHz wide. Of the 64 sub-carriers, 48 are used to transmit data and four are pilot signals, used to detect frequency offset and phase noise. The remaining 12 sub-carriers are not used in order to prevent inter-channel interference and DC offset.

Frames are transmitted as a sequence of OFDM symbols, each 4 μs long. Each OFDM symbol consists of data transmitted concurrently on the 48 data sub-carriers. The modulation scheme defines how bits are represented as waves on each sub-carrier. Four modulation schemes are used in 802.11a: BPSK, QPSK, QAM-16, and QAM-64, which can encode one, two, four, and six bits per symbol respectively. Lower modulation rates are more resistant to noise.

The physical layer uses convolutional coding, a type of forward error correction, to increase robustness to noise. The code rate expressed as m/n represents the fraction of useful information per bit transmitted. In an m/n rate code, n coded bits are transmitted for every m data bits. Lower code rates contain more redundancy and are more resistant to noise.

802.11a defines eight bit-rates, for different combinations of modulation and coding rates. The bit-rates are shown in Table 2.1.

2.3 Decoding Convolutional Codes

Several algorithms exist to decode convolutional codes. SoftRate requires a soft-output decoder; a decoder that generates reliability measures for decoded bits. Two

Bit Rate	Modulation	Code Rate
6 Mbps	BPSK	1/2
9 Mbps	BPSK	3/4
12 Mbps	QPSK	1/2
18 Mbps	QPSK	3/4
24 Mbps	QAM-16	1/2
36 Mbps	QAM-16	3/4
48 Mbps	QAM-64	2/3
54 Mbps	QAM-64	3/4

Table 2.1: 802.11a bit rates. The bit-rate is a function of the modulation scheme and the code rate.

such algorithms are implemented in Airblue: the soft output Viterbi algorithm (SOVA) and a simplification of the BCJR algorithm called Max-Log-MAP. [1] The BCJR algorithm presented in [2] is prohibitively expensive to implement in hardware. We found that the Max-Log-MAP decoder provided a good trade-off between implementation complexity and decoder performance. The results presented in this paper use the Max-Log-MAP decoder because Airblue's implementation of that algorithm produced more accurate reliability measures than Airblue's SOVA implementation.

[1] The BCJR algorithm is also referred to as the MAP algorithm

Chapter 3

Hardware Platform

This project was developed using the Airblue platform. Airblue is a flexible FPGA-based radio prototyping platform based on orthogonal frequency-division multiplexing (OFDM). This project and the Airblue platform were developed using Bluespec [6], a high-level hardware description language.

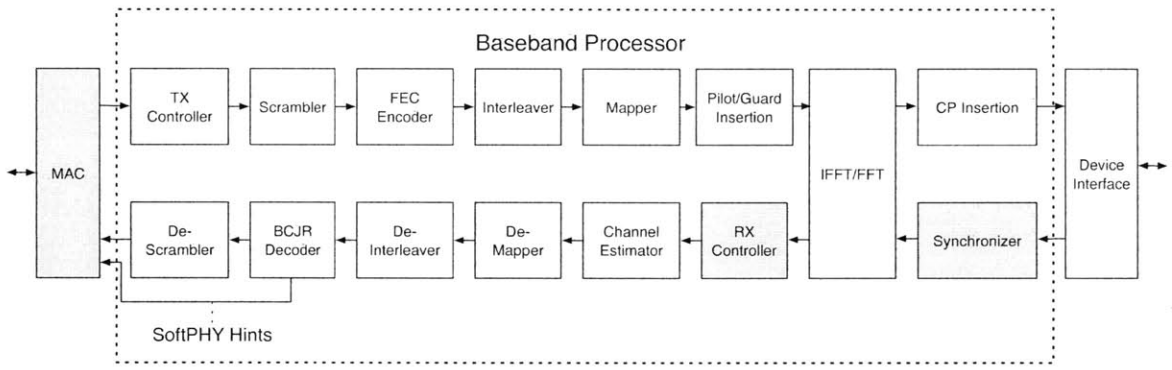


Figure 3-1: The Airblue pipeline. Highlighted modules were modified to support the SoftRate protocol.

The Airblue pipeline is shown in Figure 3. Implementing SoftRate requires modifications to the MAC layer and to the receive pipeline in the PHY layer. SoftPHY hints are extracted from the convolutional decoder and passed to the MAC layer. The RX Controller was modified to drop SoftPHY hints when the physical layer header is corrupted. To make low bit-rates useful at low SNRs, we implemented a synchronizer with improved packet detection at high noise levels.

Airblue's design makes it a good match for this project. The flexible nature of the platform allowed us to expand the MAC-PHY interface to include SoftPHY hints as well as data. Since the modules were designed in a latency-insensitive manner, we were able to replace the Synchronizer without affecting the correctness of other components. Finally, the implementing the MAC layer in hardware allowed us to compute and embed the bit-error rate feedback in 23 μs, which is within a slot time after the Short Inter Frame Spacing (SIFS) interval.

Chapter 4

Implementation

This chapter describes the implementation of the SoftRate protocol in the Airblue platform. The system consists of hardware components in the MAC and the physical layers and a software channel model for simulation.

The physical layer computes per-bit confidence hints and passes them up to the MAC. The MAC computes the channel bit-error rate by converting the confidence hints to bit-error probabilities and averaging those probabilities over the length of the packet. This bit-error rate estimate is sent as feedback to the original transmitter. If the packet is decoded without errors, the feedback is appended to an 802.11 ACK message. Otherwise, the the feedback is sent in the time slot reserved for ACK messages.

4.1 MAC-PHY interface

The interface between the MAC and the PHY layers allows decoded data to be passed up from the PHY, and data to be transmitted to be passed down from the MAC. The interface also allows the layers to communicate per-packet information, such as bit-rate and packet length.

SoftRate requires the receiving PHY to pass up per-bit confidence hints along with data to the MAC. The PHY passes up data as a sequence of bytes along with

the confidence hints for each bit in the byte as they are decoded. The confidence hints themselves are represented using 8-bit values in this project.

The 802.11a protocol has strict timing requirements for transmitting link-layer acknowledgements; the ACK must be transmitted within a slot time (9 μs) after the SIFS duration (16 μs).

If the physical layer were to wait to decode a complete packet before passing it up to the MAC, the MAC would not be able to compute the packet average bit-error rate from the confidence hints in the time alloted to transmit an ACK message.

4.2 Hint-BER table

To compute the average bit-error rate, the MAC first translates the confidence hints to bit error probabilities. In theory, the confidence hints generated by the convolutional decoder are log-likelihood ratios (LLRs); they represent the logarithm of the probability that the bit is correct divided by the probability that the bit is in error. Therefore, the probability that the bit was decoded incorrectly in terms of the confidence hint LLR is [12]:

$$p_{error} = \frac{1}{1 + e^{LLR}}$$

The relationship requires that the inputs to the convolutional decoder – the outputs of the demapper – are themselves LLRs. For demodulating BPSK symbols in the presence of additive white Gaussian noise, the LLR is proportional to the symbol amplitude and inversely proportional to the noise variance. Computing the LLRs for other constellations, such as QAM-16 and QAM-64 is more complex but [11] gives a good approximation.

Computing bit-error probability from the equation for each received bit would be too computationally expensive. Since there confidence hints are 8-bit values, we compute a 256-entry table mapping hints to probabilities ahead of time, implementing it as a look-up table in hardware.

4.3 Interference detection

The MAC differentiates between bit errors caused by signal attenuation and fading and errors caused by interfering transmissions. Lower bit rates are more robust to signal attenuation and fading. Nevertheless, the MAC should not lower the bit rate in response to collisions because that would increase the duration of each frame and conflict with other mechanisms the MAC uses to cope with interference [12]. The MAC computes packet bit-error rate feedback from only the interference-free portions of the frame.

This method requires that the MAC identify bit errors due to collisions. This project uses the technique described in [12]: the MAC identifies interference by detecting a jump in the average bit-error rate. Since the bits within a symbol are transmitted concurrently, the MAC uses the average bit-error rate over a symbol and uses the difference between symbols to detect collisions. If the difference exceeds a threshold, the MAC ignores the remainder of the frame when computing the average bit-error rate.

4.4 BER averaging

The BER averaging module computes the arithmetic mean of the bit error probabilities from the Hint-BER table. Since packets can be of variable length, computing the mean requires a division by the packet length. However, fixed-point dividers are expensive in FPGAs taking up a large number of available resources.

Furthermore, the bit error probabilities have a wide range of values ranging as small as 10^{-9} and as large as 0.5. To represent the full range of values for the maximum packet size (2304 octets [1]) requires at least 44 bits. A naive approach would require a large fixed point divider, which would use a large number of resources to realize in an FPGA. This seems like a waste since our final result requires a wide range, but not high precision.

This project explored two different strategies to avoid the need for a large divider.

4.4.1 Log Domain

The first strategy focused on the idea that divisions are easy to compute in the log-domain, since they are turned into subtraction. However, addition in the log-domain is more difficult. The Log-MAP decoder uses the Jacobi logarithm to compute the addition in the log domain [9]:

$$\ln(e^a + e^b) = \max(a, b) + \ln(1 + e^{-|a-b|})$$

We can rewrite the last term as a function of $-|a - b|$:

$$\ln(e^a + e^b) = \max(a, b) + f_c(-|a - b|)$$

The function f_c can be thought of as a correction function to the simple max. Typically, $f_c(.)$ is approximated using a pre-computed table. In [9], the authors found that a table with eight values for $|a-b|$ in the range 0 to 5 was sufficient for the Log-MAP decoder, and that a finer representation did not achieve any decoding improvement.

The Jacobi logarithm with an 8-entry table makes efficient use of FPGA resources. However, computing the running sum of the bit error probabilities results in poor accuracy. Consider a 1024-bit packet where each bit has an equal error probability of 2^{-10}. Assuming that the first 64 values sum correctly to 2^{-4}, the next sum would be computed as:

$$\begin{aligned}
\log_2(2^{-4} + 2^{-10}) &= \max(-4, -10) + \log_2(1 + 2^{-|-4+10|}) \\
&= \max(-4, -10) + f_c(6) \\
&\approx -4.0
\end{aligned}$$

$f_c(6)$ is approximately 0.02, but the smallest value we can represent with four fractional bits is larger than that: 0.0625. With this method the computed sum would be $2^{-4.0}$ rather than the actual sum of one (2^0).

Accurately computing the sum over the ten thousand or more bits in a large packet would require a much larger table and at least 13 fractional bits.

Rather than increasing the size of the table, we change the order in which we sum the probabilities. Each bit-error probability is distributed into 64 exponentially spaced buckets The first bucket contains only values $2^{-33} < x \leq 2^{-31}$, the next values $2^{-31} < x \leq 2^{-29}$ and so on. At the beginning of each packet each bucket is marked as empty. The modules uses the following algorithm:

1. Accept the next probability input.

2. If the bucket for the probability is empty, store the probability in the bucket and mark it as full.

3. If the bucket is full, compute the sum of the input and value in the bucket using the Jacobi logarithm.

   (a) If the sum falls in the same bucket, store the value in the bucket

   (b) Otherwise, mark the bucket as empty and use the sum as the next input

The algorithm has an amortized throughput of at least one probability per two cycles, since it marks a bucket empty when it does not accept new input.

4.4.2 Linear Domain

The second method computes additions in the linear domain and then converts the sum and the divisor to the log domain to perform the division. To convert to a power of two, the module counts the number of left shifts necessary such that the highest order bit is a one. The integer part of the exponent is the difference between the number of available integer bits and the number of left shifts performed minus one.

For example, the number 27.1875 as a fixed point number in binary with eight integer and four fractional bits is:

$$00011011.0011 \text{ left shift } 3 = 11011001.1000$$

The integer part of the exponent is therefore $8 - 3 - 1 = 4$. The fractional part is computed by looking up the next four bits, 1011, which correspond to 0.75. Therefore 27.1875 is approximately $2^{4.75}$.

Once both the probability sum and the number of bits are converted to exponents of 2, the module computes the average bit-error rate by taking the difference between the two exponents. This effectively divides the expected bit-errors by the number of bits.

4.4.3 Throughput

The BER averaging module uses four lookup tables to process four SoftPHY hints per cycle. Since the MAC runs at 25 Mhz, the module can achieve a throughput of up to 100 Mbps, which exceeds the highest 802.11a bit-rate of 54 Mbps.

4.5 Rate Selection

The rate selection module is responsible for choosing the bit rate at the sender based on feedback from the receiver and the frame length. The module uses the rate selection algorithm from [12]: The module contains precomputed ranges (α_i, β_i) for each rate R_i such that R_i is the optimal if the BER at R_i falls within (α_i, β_i).

The module selects the next higher rate when the BER is less than α_i, and the next lower rate when the BER is greater than β_i. The rates are precomputed for frame lengths of 4096 bits and adjusted for other frame lengths.

A frame that is twice as long requires half the BER to achieve approximately the same packet error rate. To adjust the bit rate, the module first converts the frame

length to a power of two using the technique described in 4.4.2. The module then adjusts α_i and β_i by the length exponent minus 12 (since $4096 = 2^{12}$).

For example, say (α_i, β_i) is $(2^{-21}, 2^{-14})$ for 4096 (2^{12}) bit frames. For a 512 (2^9) bit frame, (α_i, β_i) is adjusted by $9 - 12 = -3$, so the resulting range is $(2^{-24}, 2^{-17})$.

4.6 Synchronizer

The synchronizer is responsible for detecting transmissions and precise time synchronization. Since the low rates (BPSK) are most useful at low SNRs, the synchronizer must be able to function accurately at low SNRs.

The synchronizer uses the short training sequence (STS) [1] to detect packet transmissions. The STS consists of 10 repetitions of a 0.8 μs symbol. This design uses a 16-sample auto-correlation to detect the STS.

The 802.11 specification has a tolerance of ± 20 ppm for the transmitted center frequency [1]. In the 5.4 GHz band, this means the receiver can see a frequency offset of up to ± 216 KHz. The synchronizer uses auto-correlation over four symbols of the STS to do a coarse estimation of the frequency offset. Since the symbol is repeated, the rotation of the auto-correlation is proportional to the frequency offset.

The auto-correlation is fed into a CORDIC arctan module, which computes and corrects for the frequency offset.

The long training sequence (LTS) follows the STS and is 160 samples (8 μs) long beginning with a guard interval (32 samples) followed by a repeated training symbol (64 samples each). We use the LTS for fine timing acquisition and fine carrier frequency offset estimation. We use both cross-correlation and auto-correlation for fine timing acquisition.

Cross-correlation is our primary means of fine timing acquisition. The synchronizer uses cross-correlation of the sign components of the received signal with the sign components of training symbol. Using only the signs allows us to compute an entire 64-sample cross-correlation in a single clock cycle, since the sign can be represented as a single bit. (Each sample requires two bits, one for the real and one for the imaginary

component). Using only the sign components also allows us to avoid normalizing the cross-correlation based on the magnitude of the received signal.

The synchronizer keeps track of the past two largest cross-correlation computed. Timing is acquired only when the two largest peaks are 64-samples apart, corresponding to the end of the two repetitions of the LTS training symbol.

Using only this approach, the synchronizer may acquire timing too early because the guard symbol is the same as the second-half of the training symbol. To deal with this, the synchronizer checks that a 32-sample auto-correlation with a lag of 64-samples exceeds a certain threshold. This correlates the first-half of the two repetitions of the training symbol. Early synchronization is prevented because the end of the STS does not correlate well with the LTS training symbol.

4.7 Channel Simulator

Evaluating the SoftRate implementation and synchronizer requires a channel simulation capable of modelling signal attenuation, fading, noise, and multipath interference effects. The channel simulation is implemented in C++, since it does not need to be realized in hardware.

The channel model is implemented as a set of reusable filters, including an additive white Gaussian noise (AWGN) model, a linear FIR filter to model multipath fading, and a filter to model carrier frequency offset. We adapted the GNU radio Jakes' fading model from [12] for use within our channel model. Most parameters, including SNR, channel coherence time, and carrier frequency offset are configurable at runtime through environmental variables. To support repeated trials, the channel model provides functions to save and restore the internal state of the filters and the pseudo-random number generator seed.

The channel model supports both linking into the hardware simulation using Bluespec's foreign function interface and running as a separate process. When run as a separate process, the hardware simulation accesses the channel model using HAsim's remote request/response (RRR) mechanism.

The RRR mechanism allows for high-speed hardware-software co-simulation. To improve performance, we implemented a multi-threaded noise generator to take advantage of multiple processing cores. We were able to achieve simulation speeds of up to 20 Mbit/s using a Virtex-5 ACP module connected to a quad-core Xeon processor [8]. The software-only simulations typically achieved speeds of 4 Kbit/s.

Chapter 5

Simulation Results

This chapter presents the simulation results using the channel simulator described in Section 4.7. We compare our SoftRate implementation with an SNR-based protocol, which uses the Schmidl-Cox algorithm [10] to estimate SNR using the preamble. The SNR feedback is embedded in an ACK frame in the same manner as the BER feedback.

The simulation consists of two nodes connected by the channel simulator. The sender transmits as many 1500 byte UDP packets as possible to the receiver. The receiver discards any packets with an invalid frame checksum. We measure throughput in terms of the number of packets for which the sender received an acknowledgement.

In these simulations, Airblue's channel estimator was disabled because the current implementation is highly sensitive to noise. To adjust for this, the channel simulator was modified to adjust the noise variance based on the fading magnitude predicted by the Jakes' fading model. This preserves the effective SNR of each sample.

5.1 Static Channels

In this section we compare the SoftRate implementation with the SNR-based protocol in static channels with fixed noise levels. Figure 5-1 shows the results of this simulation.

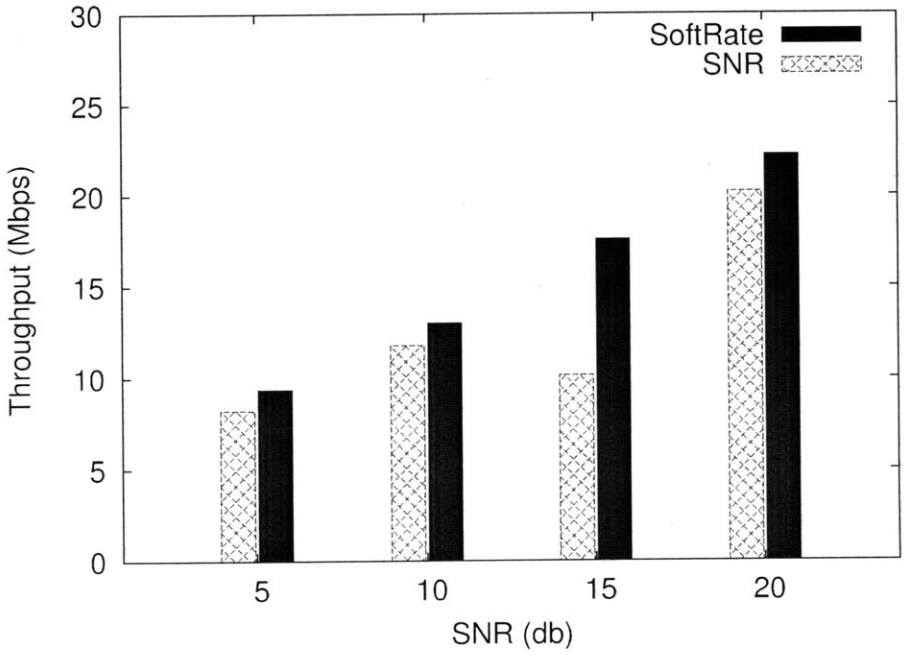


Figure 5-1: Throughput versus SNR at various noise levels. The SoftRate implementation achieves higher throughput than the SNR-based protocol. The performance gap is large at 15db because the Schmidl-Cox algorithm is less accurate at higher SNRs and therefore the SNR-based protocol is more likely to over-select the bit-rate and suffer a packet loss. At 20db, the performance gap narrows because the highest bit-rate is optimal, so over-selection is not possible.

The SoftRate implementation typically achieved 10-14% higher throughput than the SNR-based protocol. This is because the SoftRate implementation is able to more accurately estimate channel quality than the SNR-based protocol. The SoftRate implementation uses the entire packet to estimate the channel BER, while the SNR-based protocol only estimates SNR based on the preamble. The SNR-based protocol is therefore more likely to select a non-optimal bit-rate.

At an SNR of 15db, the SoftRate implementation has a performance improvement of over 73% of the SNR-based protocol. This is because the Schmidl-Cox algorithm is less accurate at high SNRs. Therefore, the SNR-based protocol is more likely to choose too high a bit-rate and suffer packet losses.

At 20db, the performance gap drops back to 10%. This is because the optimal bit-rate is 54 Mbps, the highest available rate. The SNR-based protocol still performs slightly worse because it occasionally chooses too low a bit-rate, which wastes available bandwidth.

5.2 Slow Fading Channels

Figure 5-2 shows the results of simulating the SoftRate implementation and the SNR-based protocol in slow fading channels. Three simulations were run with channel coherence times of 40 ms, 8 ms, and 4 ms. In each simulation, the sender transmits as many 1000-byte packets as fast as possible. We use the number of acknowledged packets to calculate throughput.

The SoftRate implementation achieves 20-25% higher throughput then the SNR-based protocol. Both the SNR-based protocol and the SoftRate implementation are able to adapt to the changing channel conditions. The SoftRate implementation achieves a higher throughput because it is able to better estimate channel conditions than the SNR-based protocol is, as seen in the static channel simulation.

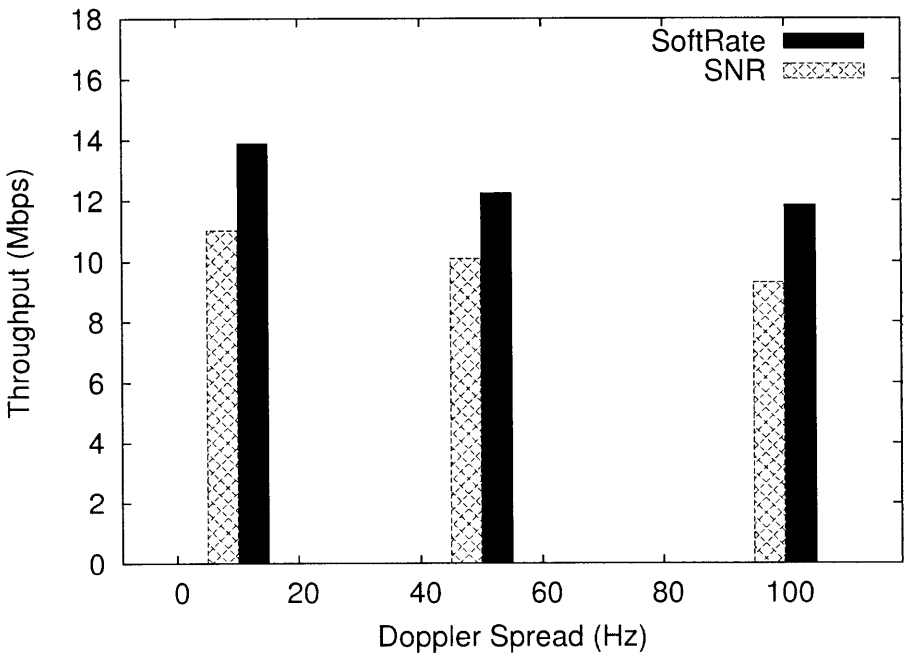


Figure 5-2: Throughput versus SNR in slow fading channels. SoftRate achieves 20-25% higher throughput than the SNR-based protocol.

5.3 Fast Fading Channels

This section presents the results of simulating the SoftRate implementation and the SNR-based protocol in fast fading channel conditions. In this experiment, the sender transmitted 1000-byte packets as fast as possible to the receiver. We simulated a fast fading channel with a coherence time of 200 μs. Figure 5-3 shows the throughput achieved by the two rate adaptation protocols as well as the throughput of five static bit-rates. The SoftRate achieved a throughput over 43% higher than the SNR-based protocol. The SNR-based protocol is likely to choose too high a bit-rate since it only measures the SNR at the beginning of a transmission. The SoftRate implementation had a throughput of 85% of the static-best bit-rate.

The fast fading case is difficult because the channel coherence time is smaller than the duration of a packet, so the fading losses experienced by sequential transmissions are uncorrelated. Furthermore, the magnitude of deep fades varies substantially between packets; some transmissions experience deep fades of -35 dB, which is likely to

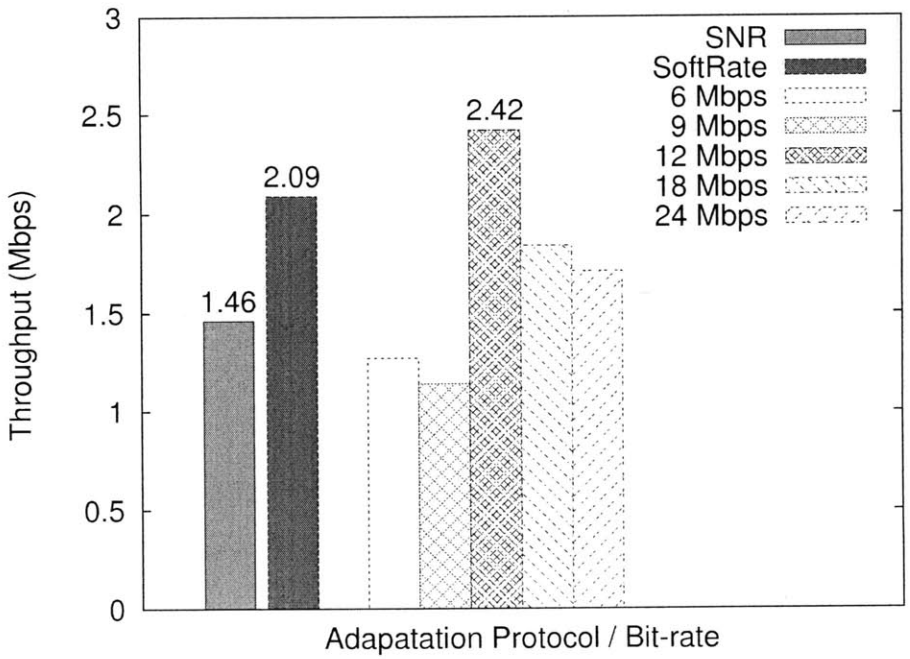


Figure 5-3: Throughput in a fast fading channels for the SoftRate implementation and SNR-based rate adaptation protocol. Also shown are the throughputs for static bit-rates from 6 Mbps to 24 Mbps.

cause a packet loss at any bit-rate, while subsequent transmissions may not experience any deep fades.

This results in a high packet-loss rate across all bit-rates. Figure 5-4 shows the packet-loss rates for the static-bit rates in the fast fading simulation. The lowest bit-rates, 6 Mbps and 9 Mbps, have a higher packet-loss rate then the 12 Mbps bit-rate because lower bit-rates have longer packet-transmission times resulting in a higher probability of experiencing a deep-fade. Since 802.11a does not use time-interleaving, the deep fades are likely to cause bit-errors and therefore packet-losses.

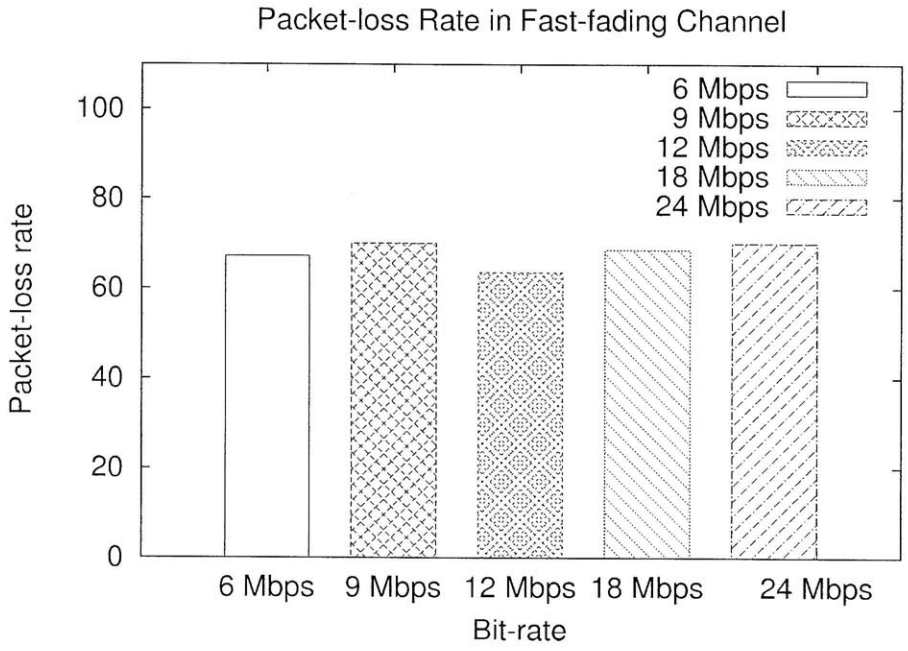


Figure 5-4: Packet-loss rate in fast fading channel. The packet-loss rate is no longer monotonic with bit-rate. In this case, 12 Mpbs is the optimal bit-rate.

Chapter 6

Conclusion

This thesis presented a hardware implementation of the SoftRate bit-rate adaptation protocol, which meets 802.11 timing requirements. The project was implemented on Airblue, an FPGA-based platform for wireless protocol development. Two methods for efficiently computing average bit-error rates in hardware were presented. We implemented an SNR-based rate adaptation protocol for comparison with the SoftRate implementation. To evaluate the rate adaptation protocols, we developed a channel simulator. This project also contributes an improved synchronizer for the Airblue platform, which more reliably detects transmissions at low SNRs. In our simulation of the hardware, we found that the SoftRate implementation achieved 10%-40% higher throughput than the SNR-based protocol across varying channel conditions. The SoftRate implementation is able to meet 802.11a timing specifications, including embedding feedback and transmitting an ACK within 25 μs of receiving a packet.

Future work includes testing the SoftRate implementation over-the air. The hardware implementation provides an opportunity to evaluate SoftRate in a real-world environment at high speeds.

Bibliography

[1] IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, pages C1 –1184, June 2007.

[2] 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.

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

[4] Joseph Camp and Edward Knightly. Modulation rate adaptation in urban and vehicular environments: cross-layer implementation and experimental evaluation. In *MobiCom '08: Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 315–326, New York, NY, USA, 2008. ACM.

[5] Gavin Holland, Nitin Vaidya, and Paramvir Bahl. A rate-adaptive MAC protocol for multi-Hop wireless networks. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 236 251, New York, NY, USA, 2001. ACM.

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

[7] Man Cheuk Ng, Kermin Fleming, Mythili Vutukuru, Samuel Gross, Arvind, and Hari Balakrishnan. Airblue: A system for cross-layer wireless protocol development. In *ANCS '10: Symposium on Architectures for Networking and Communications Systems*. (to appear).

[8] Man Cheuk Ng, Kermin Fleming, Mythili Vutukuru, Samuel Gross, Arvind, and Hari Balakrishnan. Efficient hardware implementation of bit-error rate estimates. (submitted).

[9] P. Robertson, E. Villebrun, and P. Hoeher. A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain. In *IEEE International Conference on Communications*, volume 2, pages 1009 1013 vol.2, June 1995.

[10] T.M. Schmidl and D.C. Cox. Robust frequency and timing synchronization for OFDM. *IEEE Transactions on Communications*, 45(12):1613 –1621, December 1997.

[11] F. Tosato and P. Bisaglia. Simplified soft-output demapper for binary interleaved COFDM with application to HIPERLAN/2. In *IEEE International Conference on Communications*, volume 2, pages 664 – 668, 2002.

[12] Mythili Vutukuru, Hari Balakrishnan, and Kyle Jamieson. Cross-layer wireless bit rate adaptation. *SIGCOMM Comput. Commun. Rev.*, 39(4):3–14, 2009.

[13] Mythili R. Vutukuru. *Physical Layer-Aware Wireless Link Layer Protocols*. PhD thesis, Massachusetts Institute of Technology, June 2010.