No Symbol Left Behind: A Link-Layer Protocol for Rateless Codes

Peter Iannucci, Jonathan Perry, Hari Balakrishnan, and Devavrat Shah
Massachusetts Institute of Technology
Cambridge, Mass., USA
{iannucci,yonch,hari,devavrat}@mit.edu

ABSTRACT

Recently, rateless codes have introduced a promising approach to obtaining wireless throughput higher than what is achieved by fixed-rate codes, especially over time-varying channels. Rateless codes like Raptor [26, 22], Strider [9], and spinal codes [23, 24] naturally process all the information available at the receiver corresponding to a packet, whether from one or many frame transmissions. However, a profitable deployment of rateless codes in a wireless network requires a link-layer protocol to coordinate between sender and receiver. This protocol needs to determine how much coded data should be sent before the sender pauses for feedback from the receiver. Without such feedback, an "open-loop" sender would not know when the packet has been decoded, but sending this feedback is not free and consumes a significant fraction of the packet transmission time. This paper develops RateMore, a protocol that learns the probability distribution of the number of symbols required to decode a packet (the *decoding CDF*), and uses the learned distribution in a dynamic programming strategy to produce an optimal *transmission schedule*. Our experiments show that RateMore reduces overhead by between 2.6× and 3.9× compared to 802.11-style ARQ and between 2.8× and 5.4× compared to 3GPP-style "Try-after-n" HARQ.

Categories and Subject Descriptors: C.2.1 [Network Architecture and Design]: Wireless communication

General Terms: Algorithms, Design, Performance

Keywords: Wireless, rateless, protocol, link-layer, HARQ

1. INTRODUCTION

For network applications on mobile devices to perform well, network protocols must cope with significant variations in wireless channel conditions caused by user and device movement, as well as interference from sources internal and external to the wireless network. These variations can occur even over short (sub-packet) time scales, posing a fundamental challenge for wireless network protocols.

Recent work on *rateless codes* over wireless networks has shown significant promise in achieving high performance over time-varying channels [26, 22, 6, 9, 24]. These codes are "regret-free" in the sense that the receiver never wishes that a different modulation or code rate had been chosen by the transmitter – all received symbols are equally useful and none are discarded. Under a traditional bit rate adaptation approach, the sender decides in advance on one of a pre-determined set of fixed-rate codes and modulation schemes (see §2). If the chosen rate is too aggressive, the resulting received symbols are nearly useless.

Rateless codes offer a natural way to adapt to channel variations via the *prefix property*: in a rateless code, an encoding with a higher rate is a prefix of any lower-rate encoding. Rather than discarding symbols which fail to decode, the receiver uses them again to form part of a lower-rate encoding once additional symbols arrive. No symbol is left unused in decoding a rateless message. The greater the number of symbols received, the higher the probability of a successful decoding.

Ratelessness does not excuse the sender and receiver from adapting to channel conditions. Instead, it unifies the processes of sensing conditions and reacting to them. For a good rateless code, the number of symbols required for decoding closely tracks changes in the prevalent channel conditions. This means that a rateless sender and receiver can skip the customary estimation of channel quality and focus on the central problem of estimating, based on feedback, the number of symbols to be transmitted.

A comprehensive rateless link protocol would include a mechanism for reliable feedback and a mechanism for dealing with channel contention. It would allow application-level end-to-end latency constraints to impose limits on packet aggregation, and it would provide theoretical guarantees that a suitable performance metric is optimized. It would circumscribe what the sender needs to know about the channel in order to optimize performance, and it would include the cost of conveying this information to the sender in the optimization.

In this paper, we identify efficiency as the right performance metric and show how the sender can maximize it given the feedback delay and a cumulative probability distribution function called the decoding CDF. Efficiency, given by the fraction of channel occupation time which is strictly necessary for reliable delivery, provides a code-neutral measure of protocol overhead. Our paper develops an efficiency-maximizing protocol, RateMore, and presents its implementation and evaluation over spinal codes [23], Strider [9], and Raptor codes [26, 22] on stationary and fading channels. RateMore accommodates soft single-hop latency constraints and provides reliable delivery. Multi-hop latency constraints and unreliable service classes are interesting unexplored directions.

We assume common half-duplex radios with 802.11-style framing and acknowledgments, and we consider operation above a minimum signal-to-noise ratio (SNR) such that acknowledgement packets sent at the lowest 802.11 convolutional code rate are reliable. We also assume that once a sender has contended for the medium, it uses

MobiCom'12, August 22–26, 2012, Istanbul, Turkey.

802.11's short inter-frame space (SIFS) mechanism to bypass contention during feedback hand-offs until the transaction is complete. Under these conditions, the feedback delay is known to the transmitter a priori via a calculation we review. Our analysis shows that the decoding CDF and the feedback delay are sufficient knowledge for the sender to maximize the average throughput of the link over all possible strategies of transmitting data and pausing for feedback.

The decoding CDF is not only sufficient, but it is practical to obtain. We demonstrate how the sender can obtain approximate yet satisfactory knowledge of the decoding CDF (e.g. performing within 1.57% of full knowledge) from just a handful of acknowledgments.

Our evaluation compares RateMore with two alternatives borrowed from fixed-rate coding. The first is an analogue of automated repeat requests (ARQ) fashioned after 802.11, and the second is incremental redundancy after the pattern of the 3GPP cellular standard, sending a fixed number of additional coded symbols between pauses for feedback. RateMore proceeds from a sound mathematical framework and outperforms both of these ad hoc approaches.

Our results show that RateMore achieves an efficiency of over 90% across all our experiments. Our experiments also show that RateMore reduces overhead by 2.6× to 3.9× compared to 802.11-style classical ARQ and by 2.8× to 5.4× compared to 3GPP-style "Try-after-n" hybrid ARQ. These translate to throughput improvements of up to 26% even under "static" channel conditions. Over fluctuating Rayleigh-fading channels, we show that RateMore performs within 1.57% of the ideal adaptation.

2. Related Work

2.1 Type-I HARQ

Hybrid ARQ (HARQ) systems [19] marry forward error correction with ARQ to raise the probability of successful reception. Type-I HARQ systems transmit coded messages and retransmit on failures, and are widely used in wireless standards such as Wi-Fi [14]. A common design technique is to specify several modes, or "bit rates", each suitable for a small range of channel conditions. Bit rate adaptation algorithms then choose what mode to use by picking a combination of modulation (symbol set) and code.

2.2 Type-II HARQ

Type-II HARQ allows the exchange of coded data over several round-trip interactions between the transmitter and receiver. Using more interactions has been shown to allow improvement over ARQ's performance under poor conditions, while not sacrificing performance under favorable conditions . Two approaches exist for combining transmissions. In HARQ with Chase Combining [5], the retransmission repeats parts of the original packet, which are then combined at the receiver. HARQ with Incremental Redundancy (IR) sends different coded bits in every transmission. This approach has been used successfully in the 3GPP LTE protocol [15], and a protocol has been built to achieve IR-HARQ over 802.11 networks in ZipTx [18].

Soljanin et al. discuss the design of HARQ with incremental redundancy [27]. The develop such a protocol for an ensemble of LDPC codes as well as Raptor codes. The basic idea is to transmit only as many coded symbols as required for the receiver to decode the message with high probability using the best possible maximum-likelihood (ML) decoding rule, presupposing a very high SNR. If the receiver is successful, the transmitter moves to the next message. If not, the transmitter sends the next sets of coded symbols in such a way that given what receiver has received (the transmitter knows the past channel condition at the receiver), there is a high probability of successful ML decoding if the SNR is only moderately high. This is repeated until successful decoding is achieved.

Another related work is by Anastasopoulos [1]. The primary difference arises in the fact that their work does not take feedback delays into account. They model the decoding CDF in terms of an error exponent, and assume that the underlying code cannot decode with more than some maximum number of symbols. They model the channel as a Markov chain and try to infer its state. Our approach is different both in the learning strategy (learning the decoding CDF) and in applying a dynamic programming strategy to compute the transmission schedule.

Most proposed HARQ schemes have a small number of effective rates to choose from. This is achieved using rate-compatible puncturing [11, 17, 10], where the mother code is partitioned into a series of punctured words, such that the decoder has good probability of successfully processing any prefix of the series. In comparison, rateless codes do not require such gentle constructions, and natively offer a free choice of rates.

Systems with either Type-I or Type-II HARQ schemes use bit rate adaptation algorithms. These algorithms take a *reactive* approach to combating temporal channel variations: systems passively monitor channel conditions in the form of the signal-to-noise ratio [4, 13], frame loss rate [29, 3], or bit-error rate [28, 25].

2.3 Partial Packet Recovery

Partial packet recovery schemes are protocols designed to recover packets that were received with errors. Data is divided into small fragments, and some method is used to detect which fragments were received correctly. The transmitter then only retransmits erroneous fragments. Seda [8] adds an overhead of a CRC-8 and an identifier to each block, and sends correction blocks alongside fresh blocks. PPR [16] uses confidence information from the PHY layer instead of a CRC to determine which bits need retransmission, and incorporates a dynamic algorithm to determine what feedback should be sent by the receiver. Maranello [12] again uses CRC to protect each block, but reduces overhead by only transmitting CRCs when the packet contains errors.

Partial packet recovery can be viewed as a finer-grained Type-I HARQ: the PPR system sends smaller blocks, retransmitting a block on error. The difference to non-PPR HARQ is the aggregation of several blocks onto a single transmission, eliminating the fate-sharing of bits in the non-PPR case.

2.4 Rateless Codes

Our experiments use three recently developed rateless codes—Raptor codes, Strider [9], and spinal codes [23]—so we start by summarizing the key ideas in these codes and the salient features of the implementations of these codes used in our experiments.

Raptor code. Raptor codes [26, 7], which are built on LT codes [20], achieve capacity for the Binary Erasure Channel where packets are lost with some probability. Not much is known about how close Raptor codes come to capacity for additive Gaussian noise (AWGN) channels and binary symmetric channels (BSC). However, there have been several attempts made to extend Raptor codes for the AWGN channel [22, 27, 2]. We adopt a similar construction to [22] in this paper, with an inner LT code generated using the degree distribution in the Raptor RFC [21], and an outer LDPC code as suggested by Shokrollahi [26].

Strider. Strider realizes the layered approach to rateless codes of Erez et al [6]. This approach combines existing fixed-rate base codes to produce symbols in a rateless manner. By carefully selecting linear combinations of symbols generated by the base codes, they show that the resulting rateless code can achieve capacity as the number of layers increases, provided the fixed-rate base code achieves capacity at some fixed SNR. The Strider implementation in our experiments

was built using reference code from Gudipati [9], with recommended parameters.

Spinal codes. Spinal codes [23] use a hash function to generate transmitted symbols. The rich structure obtained using the hash function can be exploited to achieve rates very close to the channel capacity, with a practical encoder and decoder. In addition to their ability to approach capacity, spinal codes also work well on short messages (256-1024 bits), making them appealing from a latency perspective.

3. Optimal Transmission Schedule

We have yet to define the decoding CDF or to show how to coordinate the sender and receiver via a *transmission schedule* derived from this CDF. This section introduces the decoding CDF and applies it to the schedule optimization problem, treating the CDF as a known quantity. We also show how to compute the feedback delay. The solution of the optimization problem proceeds from these two inputs via a dynamic programming algorithm, which we demonstrate with an example. The next section will show how to learn the CDF using feedback from the receiver.

3.1 Decoding CDF

We abstract a rateless code as an encoder which produces an infinite sequence of encoded symbols (bits or constellation points/channel usages) from a finite message, and a decoder which takes any prefix of this infinite sequence and returns either nothing or the correct message. Supposing that all input messages are protected equally, and that the channel parameters drawn from some unknown distribution, the behavior of the code on this channel is characterized by a single function giving the probability with which a message can be decoded correctly after a certain number of symbols have been received by the decoder.

We assume that this probability increases monotonically with the number of symbols received; otherwise, a better (and admittedly more expensive) decoder could attempt to decode with only the first symbol, then with the first two, and so on, guaranteeing monotonicity. If every message is eventually decoded, then the function can be viewed as the cumulative distribution for a random variable that we denote n. Changes to the code parameters, channel conditions, or code block length will affect this distribution.

The decoding CDF has three desirable features from the perspective of the protocol:

1. It succinctly captures all of the uncertainties in the system, including those due to fluctuating outcomes on a stationary channel, time-varying channel parameters, and uncertainty about these parameters. Moreover, a schedule determination algorithm that relies only on the decoding CDF is insulated from the details of the code.

2. It enables the protocol to explicitly compute, and thus maximize, the expected throughput of a transmission schedule.

3. It can be learned from receiver feedback. §4 shows how beliefs about the decoding CDF can be updated from the number of symbols needed to decode each packet. Alternatively, the CDF can be estimated using offline simulations of the behavior of the code. Either way, the sender and receiver have common knowledge of the decoding CDF.

3.2 Optimization Problem

Given a decoding CDF, we would like to obtain a rule for the sender that determines when it should transmit and when it should pause for feedback. This rule takes the form of a sequence of positive integers n_i, $i \in \{1, 2, \ldots\}$. Each n_i indicates cumulatively how many symbols should be transmitted before the sender begins its i^{th} pause.

If feedback were free (i.e., took 0 time), then a throughput-maximizing sender would pause after each symbol transmission. In reality, it takes many microseconds to turn the radios from transmit to receive mode, re-synchronize, complete any ongoing decoding operations to determine whether a positive ACK should be sent, encode the ACK, and turn the radios around again to return control to the sender. The *feedback delay* is the time between the sender's last symbol prior to a pause for feedback and the same sender's first useful symbol following the pause for feedback. With 802.11a/n timings and the most reliable ACK coding rate,

$$\begin{aligned} T_{\text{feedback}} &= \text{SIFS} + \text{preamble} + \text{ACK payload} + \text{SIFS} + \text{preamble} \\ &= 64\ \mu\text{s} + 4\ \mu\text{s} \cdot \left\lceil \frac{\text{ACK bits} + 6}{24} \right\rceil \end{aligned}$$

On a 20 MHz channel with the standard guard interval, 802.11 sends 12 million symbols per second. Thus, if the sender and receiver have only one coded transmission in play at a time, the cost of a single bit of feedback could equal the cost of 816 symbols. This number can be reduced by aggregating many packets to divide the large constant cost of feedback across more useful bits. We explore the details of such aggregation in §5. Even for Strider, the code with the largest packet size of those we considered, transmitted frames are only on the order of 3750 symbols each, so that a pause for feedback after each frame would occupy 18% of the total time spent on transmission plus feedback.

Let

$$n_f = \frac{T_{\text{feedback}}}{\text{\# aggregated packets}} \cdot 12 \text{ million } \frac{\text{symbols}}{\text{sec}}$$

This is the amortized cost of feedback in units of foregone symbols. Note that higher layers cannot tolerate an arbitrary increase in latency caused by excessive aggregation, so one can never drive the cost of feedback to zero. Suppose that n_f already takes into account as much amortization as is possible subject to higher-layer latency constraints.

We wish to send a packet whose length before any coding is b bits, after coding it using a rateless code. The decoding CDF for the rateless code specifies $\mathbb{P}(n > \cdot)$, the probability that decoding will be successful after receiving n symbols.

Consider a general transmission schedule for reliable delivery. The sender first transmits n_1 symbols, then pauses for feedback. If decoding fails, the sender transmits $n_2 - n_1$ symbols before pausing a second time. In sum, before the i^{th} pause, the sender transmits n_i symbols. We seek an assignment of values to n_1, n_2, ... that minimizes the average time spent delivering each b-bit packet.

Let

$$\begin{aligned} p_i &= \mathbb{P}(\text{first success after } i \text{ feedback rounds}) \\ q_i &= \mathbb{P}(\text{stop after } i \text{ feedback rounds}). \end{aligned}$$

These two quantities differ if the sender gives up on this packet without success and moves on to the next one. The sender will spend an average of $\sum_i q_i(n_i + i \cdot n_f)$ symbol-times on each message, including time spent transmitting and time spent waiting for feedback.

The *efficiency* of a transmission strategy is the fraction of this time *strictly necessary* for reliable delivery. Its formal definition is motivated by two observations.

1. For any feedback-based link-layer protocol, it is essential that the sender solicit feedback at least once for each network-layer packet to ensure that it has been received, so the sender can then proceed to the next packet.

2. It is necessary for the transmitter to transmit, on average, at least $\mathbb{E}[n]$ symbols. If the transmitter sends less than this number of symbols, then it follows that some fraction of packets are not decoded correctly.

Combining these two observations, we define efficiency as

$$\eta = \frac{\mathbb{E}[n] + n_f}{\sum_i q_i(n_i + i n_f)} \tag{1}$$

Note that if n_f is nonzero, the only way to achieve perfect efficiency will be for the sender to guess n correctly every time. If n has positive entropy arising from unpredictable channel variations, as in practice, we will be unable to achieve 100% efficiency, but our goal is to come as close as possible to the ideal efficiency.

Minimizing the time spent delivering a message is equivalent to maximizing η. In principle, this problem seems like a difficult multi-dimensional search, but it turns out that the optimal n_i assignments can be obtained by dynamic programming using only the decoding CDF as input. The optimal substructure in this problem is revealed by interpreting the transmitter's decisions in the framework of a dynamic game, as we explain next.

3.3 Dynamic Game Formulation

Suppose that we are playing a game against nature, and that nature chooses n from the known decoding CDF distribution but does not tell us the value. Our first move is to transmit n_1 symbols; nature's behavior is to use its hidden knowledge of n to determine whether the game ends or continues. We then transmit $n_2 - n_1$ symbols, and nature once more determines whether the game ends, and so on.

Our score at the end of the game is the negative of the total time we spent transmitting symbols or waiting for feedback (we measure the time in units of "symbol-time"). The negative is because we want the game to end as soon as possible. Because nature only acts to end the game, the optimal strategy depends only on n_f and the distribution of n (the decoding CDF), but not on any information arising during gameplay.

To develop an optimal strategy for the game, we apply backward induction. Supposing we find ourselves at some node of the decision tree where i symbols have already been transmitted, we must decide how many additional symbols j_i^* we will transmit before pausing to minimize our expected time-to-completion. For some choice of j, the corresponding expected time is

$$t_{ij} = j + n_f + \mathbb{P}(n > i + j \,|\, n > i) t_{(i+j)j^*_{i+j}}$$

That is, we pay an immediate cost of $j + n_f$, and with some probability given by the decoding CDF and expressed in the third summand above, the game will continue and we will incur additional costs according to our optimal strategy. The third term is somewhat counter-intuitive: it says that with probability $\mathbb{P}(n > i + j \,|\, n > i)$, the additional time required is $t_{(i+j)j^*_{i+j}}$; the tricky part is that the expression of t_{ij} now depends on $t_{\ell j'}$, where the index $\ell = i + j$ is *greater than* i.

We address this issue below, but for now observe that choosing

$$\begin{aligned} j_i^* &= \arg\min_{j>0} t_{ij} \\ t_i^* &= t_{ij_i^*} \text{ so that we can write} \\ t_i^* &= j_i^* + n_f + \mathbb{P}(n > i + j_i^* \,|\, n > i) t^*_{i+j_i^*} \end{aligned} \tag{2}$$

produces the optimal strategy[1].

The reason is that if we know the optimal strategy and the corresponding expected time for all $i' > i$, then we can compute the strategy and expected time for i. So, if n were bounded above by some finite value, one can use dynamic programming to find the optimal strategy to minimize the expected completion time for all i. We would then choose $n_1 = j_0^*$, $n_{i+1} = n_i + j_{n_i}^*$.

[1]The strategy is a subgame perfect equilibrium.

3.4 Finite Termination by Tail Fitting

Some finesse is required to terminate the infinite recursion onto larger and larger i. The problem is that one may easily encounter a point where there is no information available from the decoding CDF for some number of symbols, n^*, required to make progress; i.e., we have no information about $\mathbb{P}(n > n^* = i + j^*)$. What RateMore does in this situation is to revert to the best possible periodic-rate schedule given all the information known. It produces a schedule where the sender pauses after sending j^* symbols, obtaining the feedback, and then continuing, until either the packet is decoded or the sender gives up. If the packet gets decoded, the sender will have obtained information about the decoding CDF for this point in the state space, which will improve the subsequent operation of the protocol.

To determine j^*, we replace the tail of our distribution for n with an *analytic form*, which produces a stationary optimal strategy. Rearranging the terms in Equation (2) and replacing the t's with a fixed (stationary) value on both sides, we find

$$\begin{aligned} t^* &= j^* + n_f + \mathbb{P}(n > i + j^* \,|\, n > i) t^* \\ 1 - \frac{j^* + n_f}{t^*} &= \mathbb{P}(n > i + j^* \,|\, n > i) \\ &= \frac{\mathbb{P}(n > i + j^*)}{\mathbb{P}(n > i)} \end{aligned}$$

The memoryless (geometric) distribution satisfies this property. We fit a geometric tail onto our distribution and compute j^* and t^* for it to bootstrap the recursion onto finite i.

We could instead have set the complementary CDF to be zero above some finite n, but this introduces undesirable oscillations into the computed expected time and strategy variables, and does not offer any insight into what the transmitter should do if it does eventually find itself in the position of choosing a j for some i larger than this limit. With the geometric tail, the transmitter simply falls back to sending the stationary number j^* of symbols before each pause. This distinction is especially important during the process of learning the empirical CCDF, because the sender will have no data for the probability of $n > n_{\max}$, where $n_{\max}$ is the largest n we have observed so far.

For the memoryless distribution with geometric parameter $0 < \beta < 1$, we solve to obtain

$$\begin{aligned} \beta^{-j} + j \log \beta &= 1 - n_f \log \beta \\ \Rightarrow k &= \ln(k + \gamma), \text{ where} \\ k &\triangleq -j \log \beta \\ \gamma &\triangleq 1 - n_f \log \beta \end{aligned}$$

Solving iteratively,

$$\begin{aligned} k_1 &= \gamma & k_{i+1} &= \ln(k_i + \gamma) \\ j^*_{\text{tail}} &= -\frac{k_\infty}{\log \beta} & t^*_{\text{tail}} &= \frac{j^* + n_f}{1 - \beta^{j^*}} \end{aligned}$$

In practice, the iteration converges rapidly.

Another useful special case of the dynamic programming algorithm is for $n = c + g$ where c is a constant and g is geometrically distributed with parameter β. In this case, we reuse the above calculation for j^*_{tail}, but take $n_i = c + i \cdot j^*_{\text{tail}}$. That is, the first step is to send $c + j^*_{\text{tail}}$ symbols, and each subsequent step is to send

j^*_{tail} symbols. This form for the distribution of n turns out to be a reasonable practical approximation to the empirical behaviors of the spinal code and Strider.

3.5 An Example

This section will use Figure 1 to illustrate the behavior of the dynamic programming strategy using a synthetic decoding CDF for purposes of illustration. We proceed top-to-bottom, then left-to-right. The top-left plot shows a CDF and when we would pause for feedback with various delays, n_f. The mapping of colors[2] to feedback costs is shown on the right axis. The middle-left and bottom-left plots show the estimated remaining time and optimal forward strategy for the same CDF after some number of symbols have been transmitted. The circles show the best division into feedback intervals, starting at 0 and stepping forward according to the bottom-left plot.

The top-right plot shows how the optimal strategy chooses a variable level of pre-feedback confidence of decoding. For the constant-plus-geometric distribution, for example, the lines would be flat. The other two plots on the right column show how tail-fitting gives us reasonable behavior even after dozens of round-trips (which might be reasonable if the cost of feedback is low enough).

4. LEARNING THE DECODING CDF

The decoding CDF is the primary input to the dynamic programming algorithm, specifying the probability that a given number of symbols will be required by the receiver for successful decoding. These probabilities depend on current channel conditions. When the channel can be characterized by a single parameter, we can obtain CDFs for different values of that parameter using off-line simulations or experiments. For example, in the case of the additive white Gaussian noise channel (AWGN) we could obtain CDFs for a range of signal-to-noise ratio (SNR) values. In practice, however, wireless radios operate in complex environments, and we expect the channel to have too many parameters for such an approach to be practical; we expect these parameters to vary unpredictably over time; and we expect that off-line simulations will differ significantly from the actual implemented system.

A robust alternative to this approach is to *learn* the CDF on the fly: that is, to estimate the CDF based on the recent history of number of symbols required for successful decoding at the receiver. The sender always consults the strategy derived from the most recent CDF estimate. This form of online learning directly accommodates variations in channel conditions due to fading and mobility.

The most general empirical distribution for the probability of successful decoding after any number number of symbols is the multinomial distribution. Thus, a very general Bayesian approach would be to learn this multinomial distribution beginning from a Dirichlet prior. This entails maintaining a histogram over the number of symbols required for decoding so far. While straightforward, a model with such a large state-space leads to slow learning and slow adaptation to variations in the channel.

Ideally, we would like to maintain a parametric distribution with a minimal number of parameters as our surrogate for the actual CDF. With this in mind, we propose two approaches: (a) Gaussian approximation, and (b) Constant-plus-Geometric approximation. From these, we pick the Gaussian approximation, for the reasons explained below.

Gaussian approximation. Our inspection of the decoding CDFs indicates that the Gaussian distribution should be a reasonable approximation at low SNR. Maximum-likelihood (ML) estimation for the Gaussian distribution requires nothing more than computing the empirical mean and variance of the number of symbols needed for successful decoding, which can be accomplished with a few accumulators. In the face of time-varying conditions, the empirical mean and variance can be *filtered* using a moving average or a similar scheme.

We chose to use Algorithm 1 for learning a Gaussian CDF. The algorithm keeps exponentially-weighted accumulators to estimate the mean, μ, and variance, σ^2. The parameter α ranges from 0 (no memory) to 1 (unlimited memory). The averages track the input with a time constant of $1/\ln(1/\alpha)$. This scheme has two advantages over a traditional exponentially-weighted moving average of the form $y \leftarrow (1-\alpha)x + \alpha y$. First, the start-up transient dies out more quickly because we weight our initial conditions less heavily. Second, the estimator's behavior is well-defined for $\alpha = 1$: inputs are retained forever, and we recover the ML estimator.

Algorithm 1 Gaussian learning with exponentially weighted moving average. Long-term performance is not sensitive to initial values.

```
function init():
    samples ← 1
    sum ← 100 (for instance)
    sumsq ← sum^2 + 10^2 (for instance)
function learn(sample, α):
    samples ← samples · α + 1
    sum ← sum · α + sample
    sumsq ← sumsq · α + sample^2
function get_ccdf(x):
    μ = sum/samples
    σ^2 = sumsq/samples − μ^2
    return normal_ccdf(μ, σ^2, x)
```

We present results in §7 showing that under the Gaussian approximation, performance of RateMore when the true decoding CDF is not available differs by only a few percent from performance when the true CDF is available. At low SNR, the Gaussian approximation with a memory parameter of $\alpha = 0.99$ performs within 1% of the known-CDF case. We also show that very aggressive learning rates of $\alpha = 0.80$ perform very well on simulated fading channels.

Constant-plus-Geometric approximation. The Constant-plus-Geometric approximation treats the random variable n as $n = c + g$, where c is a constant and g is a geometric random variable with parameter β. Like the Gaussian, this family of distributions has two parameters, c and β. We considered this approximation because of the analogy with the presence of *error exponents* in most good codes. When a code has an error exponent, the probability of a decoding error after transmitting n symbols scales as $\exp(-\gamma n)$ for some constant γ as long as n is large enough to push the rate below the channel capacity. This minimal n corresponds to the constant summand c, and the exponential drop in error probability with increasing n corresponds to the geometric drop in probability of unsuccessful decoding with increasing g.

Additionally, if this approximation is good, the the resulting dynamic programming strategy has a simple form: for the first transmission, send a number of symbols $c + j^*_{\text{tail}}$, then pause for feedback. If the receiver has not decoded the transmission, send j^*_{tail} more symbols before each subsequent request for feedback.

Unfortunately, despite this elegance, this distribution does not have a convenient maximum-likelihood estimator. The most likely value of the constant c is upper bounded by the min of n observed

[2]Unfortunately, this chart is best viewed in color; we recognize that it is a problem when not viewed online or on a color print-out; note, however, that the different "levels" correspond to different feedback costs.

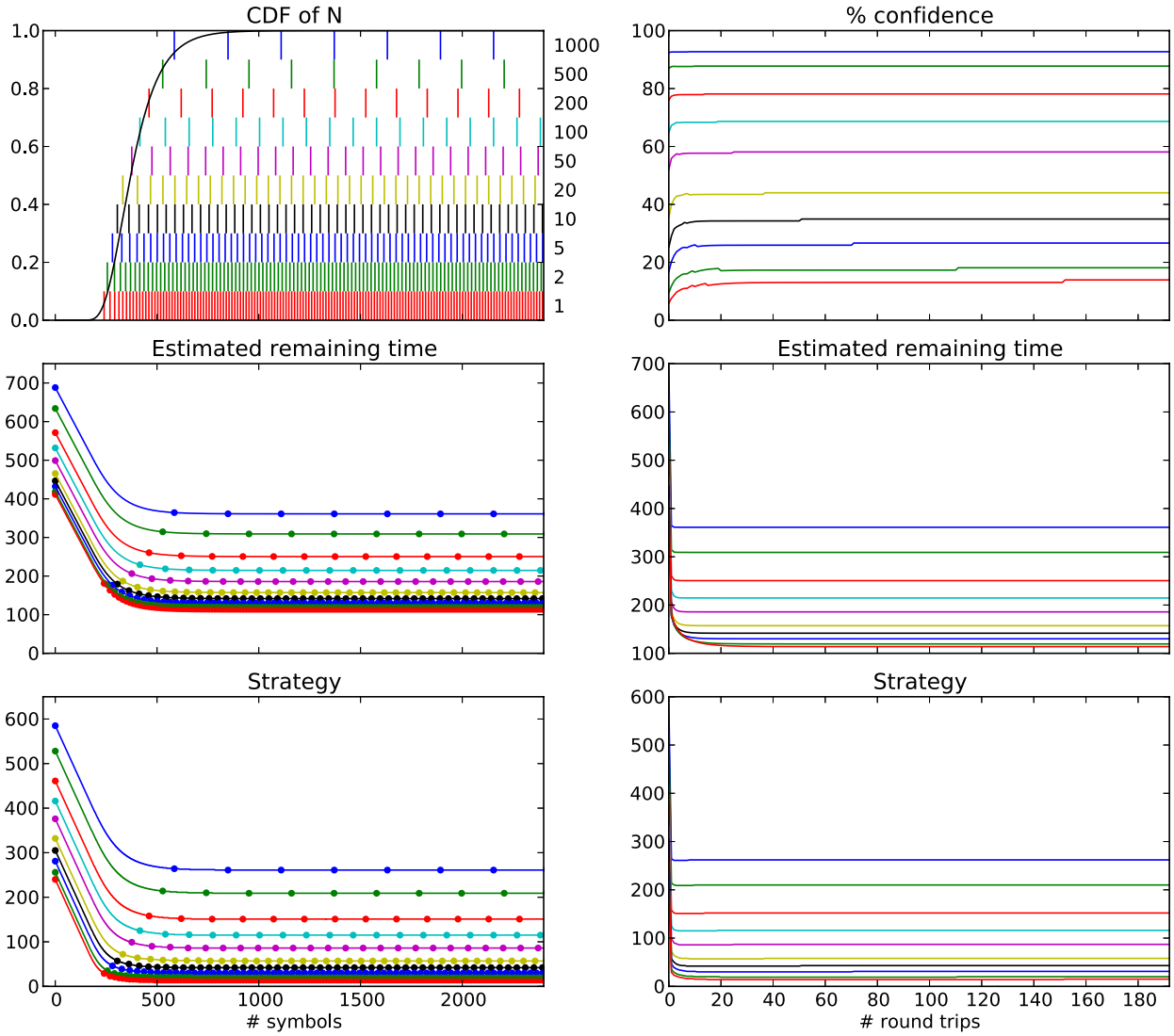

Figure 1: An illustration of the output of the dynamic programming algorithm

over all decoding attempts. In practice, this will lead the dynamic programming algorithm to under-estimate the number of symbols necessary for decoding. For this reason, we performed our experiments using the Gaussian approximation.

5. Packing Packets and Block ACKs

Rateless codes, particularly in high bandwidth situations, benefit from a method to pack symbols belonging to multiple distinct network-layer packets into the same link-layer frame, and using a "block ACK" scheme (as in 802.11e/n) to amortize the cost of sending feedback about the decoding state of each packet. Packing packets into a single frame is a useful amortization not only when the packets are small in size, but also when only a few more symbols are required to be transmitted to successfully decode the packet. As link rates increase, it is likely that any code (whether rateless or using incremental redundancy) that decodes using symbols from previous frame transmissions will benefit from packing multiple distinct packets into the same frame and using block ACKs.

If per-packet latency were not a concern, we could pack as many packets as we wish into one frame, making n_f as small as we wish. In practice, however, we care about this metric: for example, if we consider a high-definition video/audio teleconference, the maximum wireless per-packet latency might be set to 50 ms, which at a bit rate of 1.5 Mbits/s works out to $n_f =$. RateMore's dynamic programming method assumes that n_f is exogenously given by such latency requirements, and that it has already been amortized over multiple in-flight packets packed into one frame.

To efficiently pack and unpack symbols from multiple packets into a single transmitted frame, we observe that because the sender's decoding CDF is learned from the receiver's feedback, both the sender and receiver can agree on the exact CDF and hence the exact feedback schedule currently in use. The sender encodes the length of the packet (in symbols) in the Signal field as in 802.11, which is transmitted inside a 4 μs OFDM symbol. From this field, the receiver of a packet knows immediately how many symbols it will receive in all. The problem is to allocate a certain number of symbols for

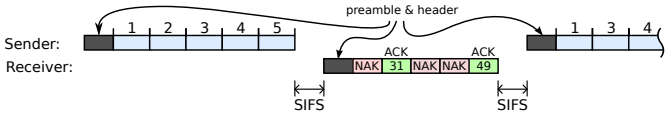

Figure 2: Symbols from many packets are aggregated into a frame in order to amortize the cost of the short inter-frame spaces (SIFS, 16µs) and preambles/headers.

packet 1, then the symbols for packet 2, and so on, as shown in Figure 2.

The solution to this problem is as follows. When the sender transmits the first frame, the receiver consults its copy of the feedback schedule to see how many symbols should have been included for a packet for which no symbols have previously been sent (obviously, this number would be the same for every packet). For example, the schedule may recommend sending 100 symbols before pausing for feedback. If a frame is received with length 500 symbols, the receiver infers that the sender wants to deliver 5 packets (as in Figure 2).

After the frame ends and the receiver finishes attempting to decode each packet, it sends an acknowledgment (ACK) frame to the sender including a short field (e.g., 6 bits) for each such packet. If the field for packet i is all ones (NAK), the packet has *not* been decoded successfully, so the sender should consult its feedback schedule (also known to the receiver) and include more symbols for that packet in the next frame. Otherwise, decoding was successful; in this case, the ACK field encodes the *fraction of this last frame's symbols for packet i that was needed to achieve a successful decoding*. Using this feedback, the sender can calculate the number of symbols that were needed to decode any given packet to estimate the decoding CDF using the learning algorithm (§4).

In general, because the sender and the receiver both agree on the updated CDF, the receiver will now know how many symbols for each unfinished packet will be in the next frame. For example, in Figure 2, three packets out of five were not decoded successfully. Each of those packets will have the *same* number of additional symbols sent in the next frame (for example, 20 symbols, according to the feedback schedule). In addition, the frame would include symbols from new packets; each of those new packets would send the same number of symbols given by the feedback schedule for new packets.

The key insight in this scheme is that we do not need to explicitly communicate the number of symbols per packet being transmitted because both parties know the schedule. This scheme works correctly if frames and ACKs are not lost, which can be engineered by ensuring that the preamble and headers are sent at low rates (one can extend this protocol to handle frame and ACK loss with some effort).

6. SOFTWARE IMPLEMENTATION

Our implementation comprises several interconnected simulations written in C++ and Python.

Codes and Channel Models. To obtain raw decoding CDFs, we performed thousands of experiments with the Spinal, Strider, and Raptor codes using the parameters shown in Table 1. The experiments produced decoding CDFs for SNRs from -5 to 35 dB on both stationary additive white Gaussian noise (AWGN) and fast-fading Rayleigh channels. The decoders had access to channel information. The slow-fading experiments used the decoding CDFs from the stationary channel, because in this regime the channel coefficients

Code	Parameter	Value
Spinal	Message size	256 bits
	k	4 bits/half symbol
	B	256 candidates/beam
	Constellation	QAM-2^{20}
Strider	K	33 layers
	M	1500 bits/layer
	Message size	50490 bits
	Outer code	1/5-rate turbo
Raptor	Message size	9500 bits
	LT degree dist.	As per RFC [21]
	LDPC design	As per Shokrollahi [26]
	LDPC rate	0.95
	Constellation	QAM-64

Table 1: Parameters for the underlying rateless codes.

fluctuate over a time scale much longer than a packet (e.g., 45 ms for a receiver moving at 10 km/hr).

To simulate slow Rayleigh fading, we directly synthesized a Gaussian process with the appropriate Doppler spectrum and used the result as a time series of channel coefficients. Noise power was maintained at a constant fraction of the average signal power. Given the Rayleigh channel coefficient and the ratio of average signal power to noise power, we computed an appropriate instantaneous SNR. For the "ideal adaptation" results in the next section, we then selected the corresponding raw decoding CDF from our empirical data by interpolating between CDFs at the nearest SNRs. RateMore itself learns the CDFs online using the Gaussian approximation. Interestingly, despite the approximation, the system's performance is as good as using the raw CDFs.

Timing. To obtain accurate values for n_f as well as throughput and latency, we built a timing model for transmission and feedback on a 20 MHz half-duplex OFDM channel designed to mimic 802.11a/n. We included the overhead associated with preamble synchronization, and assumed that all feedback was transmitted at the lowest 802.11a/n rate of 6 Mbps for reliable delivery. We did not include a simulation of contention, but we used the SIFS interval ($16\,\mu$s) to determine how long the parties would have to wait for the ACK stage.

Because the nature of our transmission schedule is to periodically incur negative acknowledgements, we wanted to ensure that in such cases the communication proceeds to successful completion before the channel is relinquished. This avoids receivers having to maintain large buffers for several ongoing conversations. Therefore, in our simulation the transmitter resumes transmitting after SIFS elapses from the receipt of an ACK frame.

Dynamic programming algorithm. Our implementation allows for the possibility of "sparse" CDFs whose values are known only for some isolated values of the number of symbols n. For the Spinal code, we found that choosing the values of n corresponding to 1/8$^{\text{th}}$-passes sacrificed virtually no performance while reducing the search space for the algorithm by a factor of roughly 8. For Strider, n should be a multiple of 3840 symbols.

The sender runs Algorithm 2 to determine how many symbols should be sent before each feedback. The points at which the CDF is known are given by x_i. The steady-state values j^*_{tail} and t^*_{tail} are computed as described in §3.4, and i_{tail} denotes the index into the x_i where recursion is terminated and the steady-state behavior takes over.

A value j_i returned from Algorithm 2 specifies that after a failed decode attempt with x_i symbols, the next decode attempt should

Algorithm 2 Given decoding CDF (sampled at x_i) and n_f, compute the transmission schedule by minimizing Expected Time To Completion (ETTC).

$j_i, t_i \leftarrow j^*_{\text{tail}}, t^*_{\text{tail}}$ for all $i \geq i_0$
for $i = i_{\text{tail}} - 1$ **to** 0 **do**
  # compute strategy after x_i transmitted symbols:
  for $j = 1$ **to** $i_{\text{tail}} - i$ **do**
    # when transmitting $x_{i+j} - x_i$ more symbols:
    $\text{ETTC}_j \leftarrow (x_{i+j} - x_i) + n_f + t_{i+j} \cdot \mathbb{P}(n > x_{i+j} \,|\, n > x_i)$
  # choose strategy that minimizes expected time:
  $j_i, t_i \leftarrow \arg\min\{\text{ETTC}_j\}, \min\{\text{ETTC}_j\}$
return $j_0, j_1, \cdots$

occur with x_{i+j_i} symbols. A sender that gets a NAK after x_i symbols should transmit $x_{i+j_i} - x_i$ more symbols before the next feedback.

Given a feedback schedule, we use Equation (1) to compute the fraction η of transmission time that is well-spent, and the overhead $1 - \eta$ that is wasted.

ARQ and Try-after-n HARQ. We implemented two alternative feedback schedules for comparison, which we refer to as ARQ and Try-after-n HARQ. With ARQ, the sender picks some number of symbols to transmit, then waits for feedback in the form of an ACK. If it receives no ACK, it drops the packet and starts over. This scheme effectively chooses a rated version of the rateless code, and does not incorporate incremental redundancy. In principle, ARQ could choose a different rate for each SNR value. However, existing ARQ systems with rate adaptation, like 802.11, do not have such a large number of rates. 802.11n with one spatial stream has only eight. We therefore require ARQ to pick eight rates to cover the SNR range from -5 to 35 dB. These eight rates are selected to maximize efficiency over all SNRs. For the Spinal code, this maximization led us to select rates of $3.4, 6.2, 11, 19, 30, 43, 56$, and 72 Mbits/s.

For Try-after-n HARQ, we picked a family of eight parameter values, as before, and sent n additional symbols of incremental redundancy before each solicitation of feedback.

7. EVALUATION

We evaluate RateMore in simulation on stationary, slow-, and fast-fading AWGN channels. Our principal results are summarized in Table 2.

7.1 Baseline comparison to ARQ, Try-after-n HARQ

Figure 3 shows the overhead of RateMore compared to ARQ and "Try-after-n" HARQ running atop spinal codes with a feedback value $n_f = 20$. These experiments are using a stationary channel simulator that introduces Gaussian noise of different variances. We run 20 trials of 100 packets at each SNR.

Across a range of SNRs, the reduction in overhead is between $2.6\times$ and $3.8\times$ relative to ARQ, and between $2.8\times$ and $5.4\times$ relative to HARQ. These reductions are significant; they translate to link-layer throughput improvements up to 26%.

7.2 Slow-fading Rayleigh channel

Using our simulated Rayleigh fading coefficients for a Doppler velocity of 10 meters per second, we compared the throughput of RateMore running with known CDFs and SNR against the throughput of RateMore with Gaussian CDF learning. Figure 4 shows a typical trace of 100 milliseconds of adaptation on a fading channel. The average SNR is 15 dB. We found that long integration times (e.g. learning parameter α close to 1) were not necessary to obtain good performance. In fact, with $\alpha = 0.8$, corresponding to an exponential

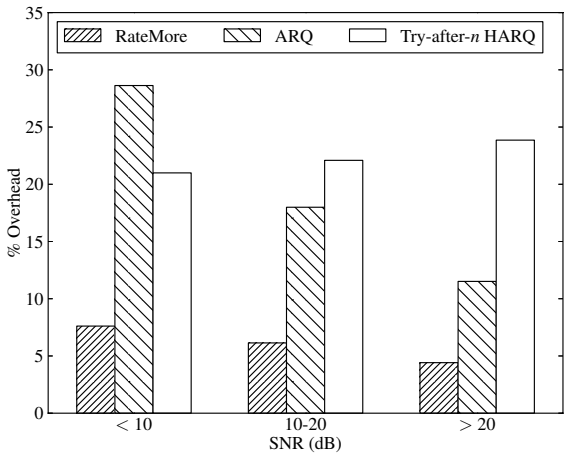

Figure 3: With spinal codes and $n_f = 20$. At low, medium, and high SNR, respectively, RateMore reduces overhead by 3.8×, 2.9×, and 2.6× relative to ARQ and 2.8×, 3.6×, and 5.4× relative to Try-after-n HARQ.

time constant of only 4.5 packets, we found aggregate throughput with learning to be within 1.57% of throughput under known CDFs and SNR. Figure 5 is a detail of Figure 4.

7.3 Efficiency

Figure 6 shows the efficiency of RateMore (we defined this metric in Equation (1)) across a range of SNRs for different values of the feedback cost, n_f. These graphs are for spinal codes (the other codes show similar results). We have grouped the data according to "low", "medium", and "high" SNRs for illustration. In general, efficiency is extremely high, always over 95% at medium and high SNRs, and over 88% for low SNR values, even when n_f is as high as 100. The conclusion is that RateMore produces a good transmission schedule across different channel conditions.

7.4 How well does learning work?

Figure 7 shows the results of experiments that evaluate how much we lose in our *learning* the decoding CDF compared to *knowing* the true CDF. We include the comparison for spinal codes, but the results are similar for other codes. At low and medium SNRs, the cost of learning, i.e., how much we lose, is a negligible 2%. At higher SNR values, the cost is always less than 6%, which is still tolerable.

The conclusion is that our Gaussian approximation of the decoding CDF, which is a simple two-parameter fit (mean and variance) works extremely well, as does the simple filtering method to estimate the mean and variance.

7.5 Streaming with a low-latency requirement

For spinal codes, Strider, and Raptor, we compared overhead and channel occupation fraction using RateMore for streaming multimedia designed to emulate a Skype voice and HD video call (Figure 8). The main constraint here is latency: we require that the number of packets in flight, times the size of a packet, should represent no more than 100 ms of audio or audio and video. The Strider and Raptor codes we used have single-packet sizes larger than this limit for the low-rate voice call. In order to meet the latency requirements, these codes would be forced to transmit partly-empty packets. Because RateMore is agnostic to the details of a specific rateless code, it enables a direct comparison of different rateless coding schemes with

Fig., §	Experiment	Result
3, §7.1	RateMore vs. ARQ, Try-after-n HARQ	Overhead reduced by up to 5.4×; throughput increased by up to 26.6%.
4, 5, §7.2	Learning under slow Rayleigh fading	Throughput only 1.57% below case of known CDFs and SNR.
6, §7.3	RateMore efficiency for various n_f	Better than 88% efficiency even when $n_f = 100$: time is well spent.
7, §7.4	Impact of learning on throughput	RateMore's learning has an impact of only 0.25%-6%.
8, §7.5	Streaming with a latency requirement	Block ACK scheduling and code agnosticism enables efficient streaming.
9, §7.6	Streaming on a fast-fading channel	Less than 15% channel occupancy even at low bandwidth.

Table 2: Summary of principal results.

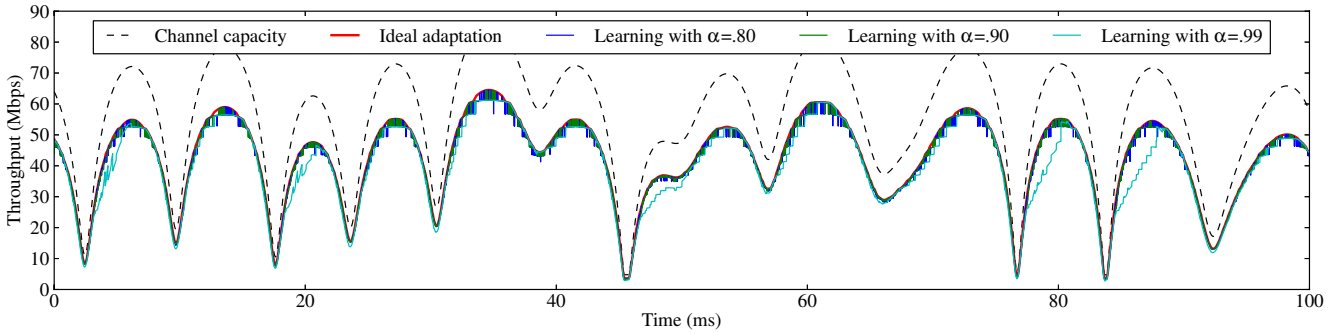

Figure 4: Learning performance of RateMore with spinal codes on a Rayleigh fading AWGN channel. Noise power is 15 dB below average faded signal power, giving an equivalent (in terms of capacity) stationary SNR of 12.8 dB. Doppler velocity is 10 m/s. The learning parameter α can be set to a very aggressive value of .8 for an aggregate throughput within 1.57% of the known-SNR "ideal adaptation" throughput. With $n_f = 10$.

different packet sizes, including for instance the effects of relatively smaller packet sizes in spinal codes.

7.6 Streaming on a fast-fading channel

When run on CDFs generated for a fast-fading Rayleigh channel, RateMore still maintains low overhead and channel occupation (Figure 9).

8. Conclusion

The recent innovations in practical rateless codes for wireless networks are an exciting development because they promise a better way to deal with the fundamental problem of time-varying channel conditions caused by mobility and interference. This paper presented the case for a different approach to link-layer protocols than the traditional methods, motivated by the property that with a rateless code, there is no obvious pause point to ask for feedback. We showed how using the decoding CDF provides a code-independent way to encapsulate the essential information of the underlying code. We developed RateMore, which incorporates a dynamic programming strategy to compute the transmission schedule, a simple learning method to estimate the decoding CDF, and a block ACK protocol to amortize feedback.

Our results show that RateMore reduces overhead by between 2.6× and 3.9× compared to 802.11-style ARQ and between 2.8× and 5.4× compared to 3GPP-style "Try-after-n" HARQ, which are the best existing deployed approaches. We demonstrated the reduced overhead of RateMore in experiments using three different rateless codes (Raptor, Strider, and Spinal codes). These significant reductions in overhead translate to good throughput improvements. For example, for spinal codes, the throughput improvement is as high as 26% compared to both ARQ and "Try-after-n" HARQ.

In conclusion, we believe that RateMore provides a practically useful link-layer protocol to coordinate between sender and receiver in wireless networks using rateless codes, improving performance when channel conditions are variable.

Acknowledgments

We thank the reviewers and Bob Iannucci for helpful comments. Two generous graduate fellowships supported this work: the Irwin and Joan Jacobs Presidential Fellowship (Perry and Iannucci), and the Claude E. Shannon Assistantship (Perry). We thank the members of the MIT Center for Wireless Networks and Mobile Computing, including Amazon.com, Cisco, Intel, Mediatek, Microsoft, and ST Microelectronics, for their interest and support.

References

[1] A. Anastasopoulos. Delay-optimal hybrid ARQ protocol design for channels and receivers with memory as a stochastic control problem. In *Communications, 2008. ICC'08. IEEE International Conference on*, pages 3536–3541, 2008.

[2] R. Barron, C. Lo, and J. Shapiro. Global design methods for raptor codes using binary and higher-order modulations. In *IEEE MILCOM*, 2009.

[3] J. Bicket. Bit-Rate Selection in Wireless Networks. Master's thesis, M.I.T., Feb. 2005.

[4] J. Camp and E. Knightly. Modulation Rate Adaptation in Urban and Vehicular Environments: Cross-Layer Implementation and Experimental Evaluation. In *MobiCom*, 2008.

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

[6] Erez, U. and Trott, M. and Wornell, G. Rateless Coding for Gaussian Channels. *IEEE Trans. Info. Theory*, 58(2):530–547, 2012.

[7] O. Etesami, M. Molkaraie, and A. Shokrollahi. Raptor codes on symmetric channels. In *ISIT*, 2005.

[8] R. Ganti, P. Jayachandran, H. Luo, and T. Abdelzaher. Datalink streaming in wireless sensor networks. In *SenSys*, pages 209–222, 2006.

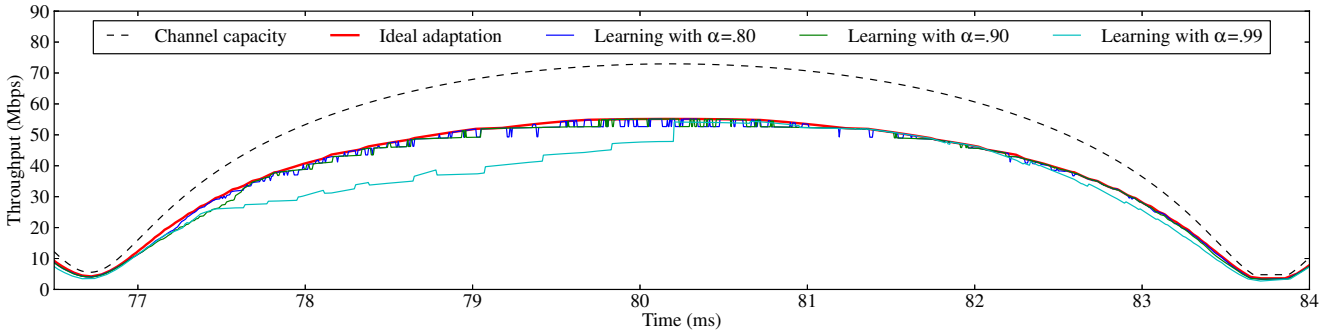

Figure 5: Detail of Figure 4.

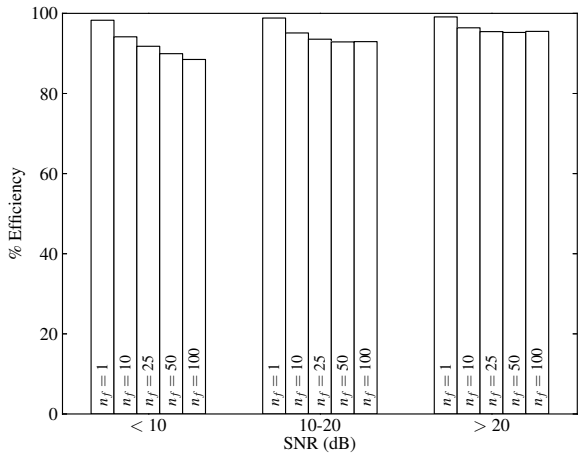

Figure 6: Even though n is random, RateMore makes good guesses and thus wastes very little time sending unnecessary symbols or waiting for unnecessary feedback.

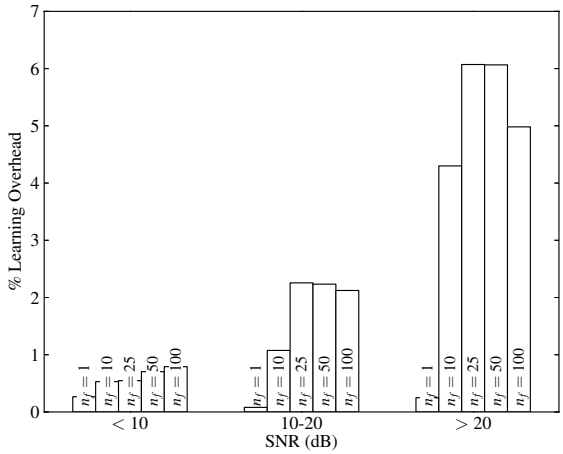

Figure 7: Gaussian learning performance on a stationary channel with spinal codes. Mean and variance are smoothed by an exponentially-weighted moving average with parameter $\alpha = .99$. Steady-state performance is comparable to performance when CDFs are fully known.

[9] A. Gudipati and S. Katti. Strider: Automatic rate adaptation and collision handling. In *SIGCOMM*, 2011.

[10] J. Ha, J. Kim, and S. McLaughlin. Rate-compatible puncturing of low-density parity-check codes. *IEEE Trans. Info. Theory*, 2004.

[11] J. Hagenauer. Rate-compatible punctured convolutional codes (rcpc codes) and their applications. *IEEE Trans. on Comm.*, 1988.

[12] B. Han, A. Schulman, F. Gringoli, N. Spring, B. Bhattacharjee, L. Nava, L. Ji, S. Lee, and R. Miller. Maranello: practical partial packet recovery for 802.11. In *NSDI*, pages 14–14, 2010.

[13] G. Holland, N. Vaidya, and P. Bahl. A Rate-Adaptive MAC Protocol for Multihop Wireless Networks. In *MobiCom*, 2001.

[14] IEEE Standard 802.11: Wireless LAN Medium Access Control and Physical Layer Specifications, Aug. 1999.

[15] J. Ikuno, M. Wrulich, and M. Rupp. Performance and modeling of LTE H-ARQ. In *Proc. International ITG Workshop on Smart Antennas (WSA 2009), Berlin, Germany*, 2009.

[16] K. Jamieson and H. Balakrishnan. PPR: Partial Packet Recovery for Wireless Networks. In *SIGCOMM*, 2007.

[17] J. Li and K. Narayanan. Rate-compatible low density parity check codes for capacity-approaching ARQ scheme in packet data communications. In *Int. Conf. on Comm., Internet, and Info. Tech.*, 2002.

[18] K. C. Lin, N. Kushman, and D. Katabi. ZipTx: Exploiting the Gap Between Bit Errors and Packet Loss. In *MobiCom*, 2008.

[19] C. Lott, O. Milenkovic, and E. Soljanin. Hybrid ARQ: theory, state of the art and future directions. In *2007 IEEE Information Theory Workshop on Information Theory for Wireless Networks*, pages 1–5. IEEE, July 2007.

[20] M. Luby. LT codes. In *FOCS*, 2003.

[21] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer. Raptor Forward Error Correction Scheme for Object Delivery. RFC 5053 (Proposed Standard), Oct. 2007.

[22] R. Palanki and J. Yedidia. Rateless codes on noisy channels. In *ISIT*, 2005.

[23] J. Perry, H. Balakrishnan, and D. Shah. Rateless spinal codes. In *HotNets-X*, Oct. 2011.

[24] J. Perry, P. A. Iannucci, K. E. Fleming, H. Balakrishnan, and D. Shah. A rateless wireless communication system using spinal codes. In *SIGCOMM*, Aug. 2012. To appear.

[25] S. Sen, N. Santhapuri, R. Choudhury, and S. Nelakuditi. AccuRate: Constellation-based rate estimation in wireless networks. *NSDI*, 2010.

[26] A. Shokrollahi. Raptor codes. *IEEE Trans. Info. Theory*, 52(6), 2006.

[27] E. Soljanin, N. Varnica, and P. Whiting. Incremental redundancy hybrid ARQ with LDPC and Raptor codes. *IEEE Trans. Info. Theory*, 2005.

[28] M. Vutukuru, H. Balakrishnan, and K. Jamieson. Cross-Layer Wireless Bit Rate Adaptation. In *SIGCOMM*, 2009.

[29] S. Wong, H. Yang, S. Lu, and V. Bharghavan. Robust Rate Adaptation for 802.11 Wireless Networks. In *MobiCom*, 2006.

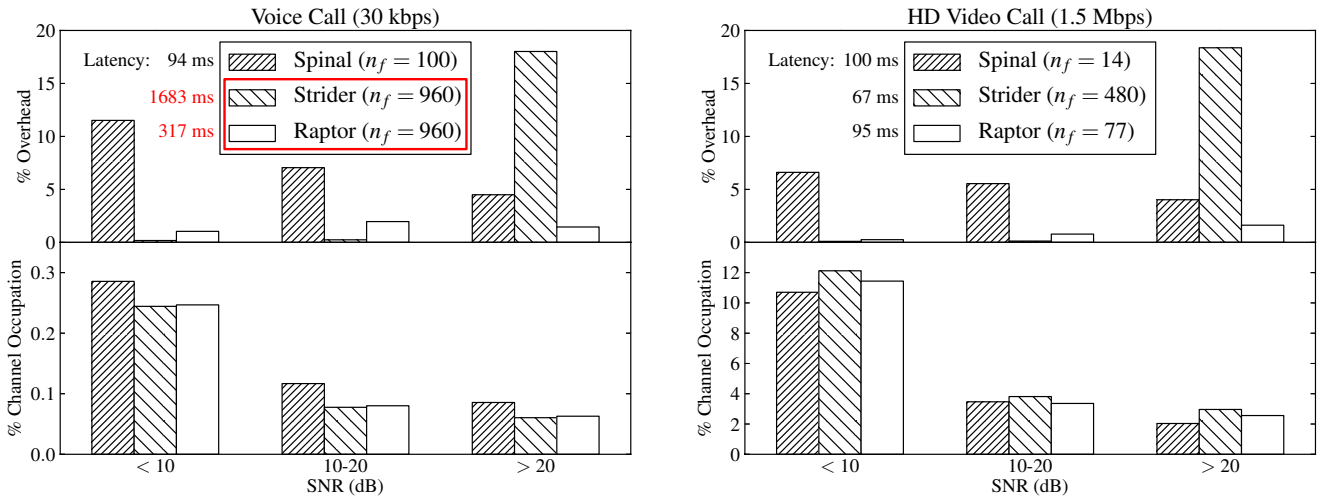

Figure 8: RateMore is agnostic to the details of a specific rateless code, and achieves low overhead in latency-critical environments. n_f has been normalized according to the largest number of parallel streams compatible with a given bandwidth-latency product. Cases are shown for a minimum-quality Skype voice call and for an HD video call. For an ideal user experience, latency should be less than 100 ms. Strider and Raptor require relatively large packets, which prevents the use of aggregation on the voice call – in fact, a single packet contains more than 100 ms of audio, as shown in red. In order to meet the latency target at the expense of channel occupation, packets could be fired off partly empty.

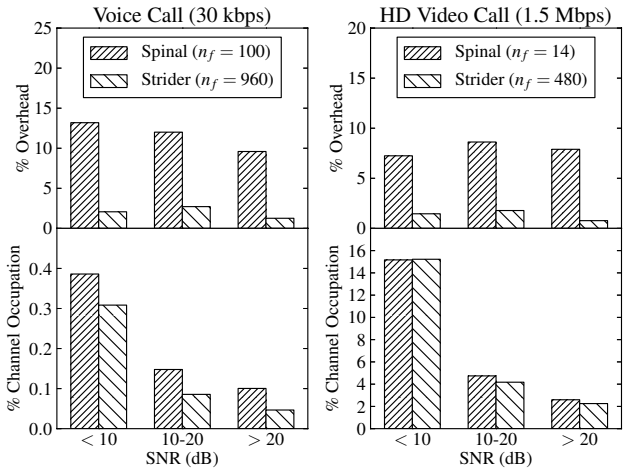

Figure 9: RateMore overhead and channel occupation on fast-fading Rayleigh channels. n_f has been normalized as in Figure 8