Spinal Codes

Jonathan Perry, Peter A. Iannucci, Kermin E. Fleming, Hari Balakrishnan, and Devavrat Shah
Massachusetts Institute of Technology, Cambridge, Mass., USA

ABSTRACT

Spinal codes are a new class of rateless codes that enable wireless networks to cope with time-varying channel conditions in a natural way, without requiring any explicit bit rate selection. The key idea in the code is the sequential application of a pseudo-random hash function to the message bits to produce a sequence of coded symbols for transmission. This encoding ensures that two input messages that differ in even one bit lead to very different coded sequences after the point at which they differ, providing good resilience to noise and bit errors. To decode spinal codes, this paper develops an approximate maximum-likelihood decoder, called the *bubble decoder*, which runs in time polynomial in the message size and achieves the Shannon capacity over both additive white Gaussian noise (AWGN) and binary symmetric channel (BSC) models. Experimental results obtained from a software implementation of a linear-time decoder show that spinal codes achieve higher throughput than fixed-rate LDPC codes [11], rateless Raptor codes [33], and the layered rateless coding approach [8] of Strider [12], across a range of channel conditions and message sizes. An early hardware prototype that can decode at 10 Mbits/s in FPGA demonstrates that spinal codes are a practical construction.

CATEGORIES AND SUBJECT DESCRIPTORS

C.2.1 [**Network Architecture and Design**]: Wireless communication

GENERAL TERMS

Algorithms, Design, Performance

KEYWORDS

Wireless, rateless, channel code, capacity, practical decoder

1. INTRODUCTION

Signal attenuation, noise, multipath fading, and interference all make it difficult to achieve high throughput over wireless networks. Achieving high throughput is challenging even when the channel is characterized by a single invariant parameter such as the noise variance or the bit-error rate, but in practice, mobility and interference cause conditions to vary over multiple time-scales. Currently deployed solutions to this problem in wireless LANs and cellular networks are *reactive*; they measure the channel to dynamically select a "bit rate"—i.e., a modulation scheme, channel code, and code rate—from a set of pre-defined choices.

An alternate approach is to use a *rateless code* between the sender and receiver [8, 12, 27]. With a rateless code, the sender encodes the message bits so that every achievable higher rate is a prefix of achievable lower rates. The sender keeps transmitting coded data until the receiver informs the sender that it has correctly decoded all the data (or the sender gives up). An ideal rateless code can be decoded correctly with modest computational complexity as soon as the effective rate drops below the capacity of the channel; the prefix property of such a code eliminates the need for the heuristics used in explicit bit rate selection.

This paper presents the encoding, decoding, and performance of *spinal codes*, a new class of rateless codes. Spinal codes can encode message bits directly to constellation symbols (which is somewhat unusual), or they can produce a sequence of coded bits to be transmitted using any pre-existing symbol set. The first approach is preferable because the spinal decoder can extract information from the raw received symbols without a demapping step, permitting the use of the same dense constellation at all signal-to-noise ratios (SNR). Even without control over the physical layer, spinal codes may be useful over an existing physical layer modulation to improve throughput and error resilience.

Spinal codes apply a *hash function* sequentially to successive portions of the original message bits to produce a pseudo-random mapping from message bits to coded bits, and then use a *constellation mapping function* to produce a sequence of symbols for transmission. Thanks to the sequential application of the hash function, two input messages differing in even a single bit result in independent, seemingly random symbols after the point at which they differ: any difference between messages is magnified in the encoded symbols. This property makes the code robust to noise and errors; it achieves reasonable rates even at SNR as low as -5 dB.

No previous code uses a hash function because good hash functions, by design, do not possess a simple, invertible structure. Therefore, unlike previous codes such as low-density parity check (LDPC) [11], LT [21], or Reed-Solomon [42] codes whose decoders exploit graphical or algebraic properties, spinal decoding requires a different strategy.

So how does one decode spinal codes? Our solution draws its strength from the sequential structure of the encoding. The idea is to search over a tree, breadth first, with edges weighted by the difference between the actual received symbols and the result of *replaying the encoder* for different potential message bit sequences. The shortest path is an exact maximum-likelihood (ML) decoding, but the size of the tree is exponential in the message length. We introduce a polynomial-time method, called the *bubble decoder*, which prunes the tree to produce an approximate ML estimate of the message. Theoretically, the bubble decoder achieves capacity. To the best of our knowledge, spinal codes are the first rateless codes with an efficient encoder and decoder that achieve capacity over both AWGN and BSC (bit-flip) models.

Our experiments conducted using a *linear-time* spinal decoder compare spinal codes to Raptor [33, 26] over a dense QAM-256 constellation, Strider [12], and LDPC codes [11]. We present four principal experimental results. First, the rate improvements of spinal codes over Raptor and Strider (both rateless codes) for AWGN channels are significant:

SNR	Raptor/QAM-256	Strider
High (>20 dB)	21%	40%
Medium ($10-20$ dB)	12%	25%
Low (<10 dB)	20%	32%

SIGCOMM'12, August 13–17, 2012, Helsinki, Finland.

Second, spinal codes outperform the best envelope of the 802.11n family of LDPC codes, because of a "hedging" effect that we identify. Rateless codes over modest code block lengths are able to take advantage of channel variations that are inevitable even when the SNR is fixed, unlike fixed-rate codes.

Third, on a Rayleigh fading channel model, spinal codes outperform Strider by between 19% and 28% at an SNR of 10 dB and by between 28% and 33% at an SNR of 20 dB. We augmented Strider with a puncturing strategy, which improved Strider's performance. We also found that spinal codes perform reasonably well even when the decoder does not have detailed or accurate fading information, unlike Strider.

Fourth, for small packets, typical of Internet telephony or gaming applications, spinal codes outperform Raptor by between $14\% - 20\%$, and Strider by between $2.5\times$ and $10\times$.

To demonstrate that spinal codes are indeed practical to implement, we have built a hardware prototype that achieves over-the-air rates comparable to equivalent software simulations for SNR between 2 dB and 15 dB. The FPGA decoder runs at 10 Mbits/s. Using the appropriate tools, we estimate that a silicon implementation of the design would operate at about 50 Mbits/s.

We believe these experimental results show that spinal codes are a promising and practical advance for wireless networks. Moreover, from a conceptual standpoint, spinal codes present a framework for making Shannon's random coding ideas, which form the basis of many fundamental capacity proofs, practical.

2. Related Work

Current wireless networks, including 802.11 and various wide-area cellular wireless standards, provide a large number of physical layer (PHY) configurations, including various codes, various parameters for these codes, several choices of symbol sets (i.e., constellations) over which to modulate bits, and a way to map groups of bits to symbols (e.g., Gray code). These networks implement *explicit*, *reactive* bit rate adaptation policies to dynamically select and configure the discrete choices and parameters provided by the PHY [15, 18, 41, 30, 43, 7].

In recent years, there has been strong interest in rateless codes over both erasure (packet loss) and wireless (AWGN and BSC) channel models. By "rateless", we mean a code where the sequence of coded bits (or symbols) when the code achieves a higher rate is a *prefix* of the sequence when the code achieves a lower rate [8]. This prefix property allows a decoder to process coded data incrementally until successful decoding is possible. Shannon's random codebook approach used in the proofs of capacity is inherently rateless, achieving capacity for channels characterized by a single noise or error parameter [32, 34]. Unfortunately, it is computationally intractable.

The desire for computationally efficient, capacity-achieving rateless codes led to Shokrollahi's work on Raptor codes [33, 9], which are built on Luby's LT codes [21]. They achieve capacity for the Binary Erasure Channel where packets are lost with some probability. On AWGN and BSC models (which model wireless better), not much is known about how close Raptor codes come to capacity. There have, however, been several attempts made to extend Raptor codes to the AWGN channel [26, 35, 4]; we compare spinal codes with an extension of the method of Palanki and Yedidia [26].

Erez et al. recently proposed a "layered approach" to design rateless codes for the AWGN channel [8]. 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. Strider [12] uses this layered approach, with a base turbo-code [6].

In contrast, spinal codes are not layered codes; they do not rely on existing fixed-rate base codes. Unlike Strider, which takes an existing fixed-rate code and symbol set system and makes modifications to the lowest PHY procedures to achieve linear combinations of symbols, the construction of spinal codes provides a single (arguably simpler) mechanism to overcome channel impediments. Spinal codes also naturally extend to the BSC case, whereas the layered approach does not. We compare spinal codes with Strider in §8.

Unlike most existing practical codes, spinal codes are nonlinear; i.e., the coded symbols (bits) are not linear combinations of input message bits. Using hash functions, it produces good coded sequences without requiring complicated operations such as multiplying message bits by a random matrix or using complex graph structures. Raptor and LT codes use a pseudo-random number generator (through choice of the graph structure) to choose which bits to XOR together, whereas spinal codes use a pseudo-random number generator directly to produce symbols.

The M-algorithm [1, 17, 28] is a method to efficiently decode over a tree for random convolutional codes [40]. Our bubble decoder may be viewed as a generalization of the classical sequential decoding algorithm as well as the M-algorithm, as explained in §4.3.

We also note a superficial similarity between the direct coding to symbols used in spinal codes and Trellis Coded Modulation (TCM) [38, 37]. TCM was crafted specifically to achieve high minimum distance between codewords under a sparse constellation for convolutional codes, whereas spinal codes aim to attain higher mean distance, obviating the need for sparse constellations. TCM is not rateless, does not achieve capacity for AWGN, is not workable (in any obvious way) for BSC, and is specific to convolutional codes.

Among fixed-rate codes for communication applications, convolutional codes [40], LDPC [11] and turbo-codes [6] are the most widely used. Because LDPC and turbo-codes perform well, much simulation work has been done on puncturing these codes and combining them with incremental redundancy in an attempt to emulate rateless operation [23, 20, 13, 31]. We compare spinal codes to LDPC codes decoded using a strong belief propagation decoder.

3. Encoding Spinal Codes

This section describes the encoder for spinal codes. We describe it in the context of a system that has full control of the physical layer, so the encoder produces a sequence of symbols for transmission and the decoder operates on the received symbol sequence to produce an estimate of the original message bits. By slightly modifying the encoder and decoder, it is straightforward to apply the code to a system that has an existing mapping from (coded) bits to symbols.

The encoding procedure takes the input message bits, $M = m_1 m_2 \ldots m_n$, and produces a sequence of symbols on the I-Q plane. At the receiver, the PHY receives a stream of symbols on the I-Q plane. The decoder processes this stream sequentially, continuing until the message is successfully decoded, or until the sender (or receiver) gives up, causing the sender to proceed to the next message. In practice, a single link-layer frame might comprise multiple coded messages, as explained in §6.

When used in rateless mode, spinal code encoder can produce as many symbols as necessary from a given sequence of message bits, and the sequence of coded bits or symbols generated at a higher code rate is a prefix of that generated at all lower code rates.

3.1 Spine Construction

At the core of the spinal code is a hash function, h, and a pseudo-random number generator, RNG, known to both the transmitter and receiver. h takes two inputs: (1) a ν-bit state and (2) k message bits.

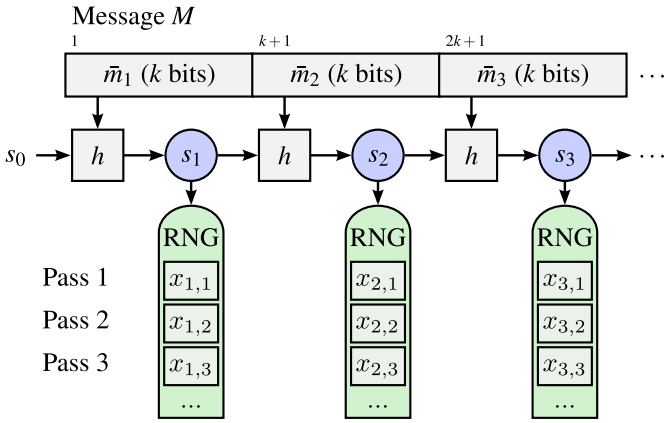


Figure 1: Encoding process. Start with a hash function, h. Compute spine values $s_i = h(s_{i-1}, \bar{m}_i)$. Seed RNG with s_i. For pass ℓ, map c bits from RNG to symbol $x_{i,\ell}$.

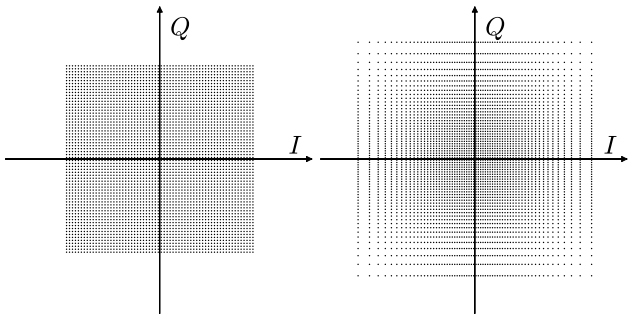


Figure 2: Uniform (left) and truncated Gaussian (right) constellation mapping functions. Same average power; $c = 6$; truncated Gaussian with $\beta = 2$.

It returns a new v-bit state. That is,

$$h : \{0,1\}^v \times \{0,1\}^k \to \{0,1\}^v.$$

The initial value, s_0, of the v-bit state is known to both the encoder and decoder, and may be considered (for now) to be the string 0^v without loss of generality.

As shown in Figure 1, the idea is to build a *spine* of v-bit states by sequentially hashing together groups of k bits from the input message. We denote bits $m_{ki+1} \ldots m_{k(i+1)}$ as $\bar{m}_i$, so the sequence of states is simply

$$s_i = h(s_{i-1}, \bar{m}_i),\ s_0 = 0^v.$$

Each of these n/k states, or *spine values* (n being the number of bits in the input message), is used to seed a random number generator, RNG. Each RNG generates a sequence of pseudo-random c-bit numbers, which are converted into output symbols using a constellation mapping function (§3.3). RNG is a deterministic function from a v-bit seed and an index to a c-bit number:

$$\text{RNG} : \{0,1\}^v \times \mathbb{N} \to \{0,1\}^c.$$

The sequence of states computed by repeatedly applying h is superficially similar to a traditional convolutional encoding, but there are three key differences. First, the hash function has a richer pseudo-random (and generally nonlinear) structure and operates on a significantly larger v-bit state, where v is on the order of 32. (Hash collisions are a potential concern; §8.5 shows that they can be made extremely rare.) Traditional convolutional codes update their state according to a linear (exclusive-or) function. The larger state space of the spinal encoder gives rise to the second major difference: the "constraint length" of this encoding goes all the way back to the start of the message, because the state at the end of any stage depends on all the input message bits in the message until that point. The third key difference is that, whereas a convolutional encoder has a constant ratio of the number of input to output bits (i.e., a fixed rate), the spinal code is rateless because one can generate as many transmission symbols as desired using the RNG. h and RNG together allow the spinal encoding to not only achieve good separation between codewords, but also ratelessness.

3.2 Hash Function and RNG

We choose h uniformly based on a random seed from a *pairwise independent* family of hash functions $\mathscr{H}$ [24]. This property guarantees that for two distinct hash inputs x and y, every pair of output values a and b is equally likely. This property is standard and attainable in practice. The encoder and decoder both know h, RNG, and the initial value s_0; if s_0 is chosen pseudo-randomly, the resulting symbol sequence is pseudo-random, providing resilience against "bad" or adversarial input message sequences (one may view the use of a pseudo-random s_0 as analogous to a scrambler).

Because our requirements for RNG are similar to those for h, one suitable choice for RNG is to combine h with a v-to-c-bit shift register.

3.3 Rateless Symbol Generation

The output of the encoder is delivered in a series of *passes* of n/k symbols each, as depicted in Figure 1. The encoder produces symbols $x_{i,1}$ for the first pass, where $x_{i,1}$ is the output of a deterministic *constellation mapping function* acting on the first c-bit number generated by the i^{th} RNG (seeded by s_i). It produces symbols $x_{i,\ell}$ for subsequent passes by generating additional outputs from each of the random number generators. The encoder continues to loop back and generate additional symbols until the receiver manages to decode the message or the sender or receiver decides to give up on the message.

Let b be a single c-bit input to the constellation mapping function. For the BSC, the constellation mapping is trivial: $c = 1$, and the sender transmits b. For the AWGN channel (with or without fading), the encoder needs to generate I and Q under an average power constraint. The constellation mapping function generates I and Q independently from two separate RNG outputs of c bits each.

We examine the two constellation mappings shown in Figure 2. The first is *uniform*, and the second produces a *truncated Gaussian* via the standard normal CDF, Φ. In terms of the average power P,

$$\begin{array}{ll} \text{Uniform: } & b \to (u - 1/2)\sqrt{6P} \\ \text{Gaussian: } & b \to \Phi^{-1}(\gamma + (1-2\gamma)u)\sqrt{P/2} \end{array} \qquad u = \frac{b + 1/2}{2^c}$$

where $\gamma \equiv \Phi(-\beta)$ limits the Gaussian's range to $\pm\beta\sqrt{P/2}$. β controls the truncation width. Very small corrections to P are omitted.

4. Decoding Spinal Codes

In this section, we present an efficient *bubble decoder* for spinal codes. This is an approximate ML decoder whose time and space complexity are polynomial in the number of bits in the message being recovered. Later in this section and in Appendix 9, we show that the polynomial-time approximation of ML decoding achieves capacity over both the AWGN and BSC models. In §8, we show experimentally that a linear-time bubble decoder achieves throughput close to the Shannon limit, outperforming state-of-the-art rated codes (LDPC) and recent rateless codes (Raptor and Strider).

4.1 The Problem

The central concept in ML spinal decoding is to search for the encoded message that differs least from the received signal. Given a vector of observations $\bar{y}$ and an encoder function $\bar{x}(M)$ that yields the vector of transmitted symbols for a message M, the ML rule for the AWGN channel is then

$$\hat{M} \in \underset{M' \in \{0,1\}^n}{\arg\min} \|\bar{y} - \bar{x}(M')\|^2. \quad (1)$$

That is, the receiver's estimated message $\hat{M} \in \{0,1\}^n$ is the one that produces an encoded vector $\bar{x}(\hat{M})$ closest to $\bar{y}$ in ℓ_2 distance. For the BSC, the only change is to replace the ℓ_2 Euclidean distance with Hamming distance.

Because spinal codes are rateless, the lengths of vectors $\bar{x}$ and $\bar{y}$ increase as the transmitter sends more symbols through the channel. Suppose that the transmitter has sent N symbols up to the present. The set of all transmitted words $\{\bar{x}(M')$ for all $M' \in \{0,1\}^n\}$ forms a dense cloud in N-dimensional space. Under Gaussian noise, nearby points in the cloud are initially indistinguishable at the receiver. As N increases, however, the average separation of the points increases. Eventually, n/N (whose dimensions are bits/symbol) drops below the Shannon capacity. The separation of the points then becomes great enough that the correct message is the arg min, and decoding terminates.

The brute-force approach to ML decoding is to conduct an exhaustive search over all 2^n possible messages. Thus, the key question is whether it is possible to develop a practical and implementable spinal decoder. Fortunately, the sequential structure of spinal codes and the powerful mixing effect of h and RNG enable efficient decoding, as explained next.

4.2 Decoding over a Tree

Because the spinal encoder applies the hash function sequentially, input messages with a common prefix will also have a common spine prefix, and the symbols produced by the RNG from the shared spine values will be identical. The key to exploiting this structure is to decompose the total distance in (1) into a sum over spine values. If we break $\bar{y}$ into sub-vectors $\bar{y}_1, \ldots, \bar{y}_{n/k}$ containing symbols from spine values s_i of the correct message, and we break $\bar{x}(M')$ for the candidate message into n/k vectors $\bar{x}_i(s_i')$, the cost function becomes:

$$\|\bar{y} - \bar{x}(M')\|^2 = \sum_{i=1}^{n/k} \|\bar{y}_i - \bar{x}_i(s_i')\|^2. \quad (2)$$

A summand $\|\bar{y}_i - \bar{x}_i(s_i)\|^2$ only needs to be computed once for all messages that share the same spine value s_i. The following construction takes advantage of this property.

Ignoring hash function collisions (discussed in §8.5), decoding can be recast as a search over a tree of message prefixes. The root of this *decoding tree* is s_0, and corresponds to the zero-length message. Each node at depth d corresponds to a prefix of length kd bits, and is labeled with the final spine value s_d of that prefix. Every node has 2^k children, connected by edges $e = (s_d, s_{d+1})$ representing a choice of k message bits $\bar{m}_e$. As in the encoder, s_{d+1} is $h(s_d, \bar{m}_e)$. By walking back up the tree to the root and reading k bits from each edge, we can find the message prefix for a given node.

To the edge incident on node s_d, we assign a *branch cost* $\|\bar{y}_d - \bar{x}_d(s_d)\|^2$. Summing branch costs on the path from the root to a node gives the *path cost* of that node, equivalent to the sum in (2). The ML decoder finds the leaf with the lowest cost, and returns the corresponding complete message.

The sender continues to send successive passes until the receiver determines that the message has been decoded correctly. The receiver stores all the symbols it receives until the message is decoded correctly. For instance, if six symbols have been received so far for each spine value, then the vectors $\bar{x}_d$ in the cost computation above have six components.

4.3 Bubble Decoding: Pruning the Tree

Suppose that M and M' differ only in the i^{th} bit. Comparing $\bar{x}(M)$ with $\bar{x}(M')$, we find that symbols from spine values $\lfloor i/k \rfloor, \ldots, n/k$ in the two transmissions are completely dissimilar. If M is the correct decoding, then M' will have a larger path cost than M. The gap will be largest when i is small. This observation suggests a helpful intuition, which can be derived more formally from the proof of the theorem in the appendix: alternate codewords with comparable path costs to the ML decoding *differ only in the last* $O(\log n)$ *bits*.

Building on this idea, suppose that we have constructed the entire ML decoding tree and computed the path costs for all of the leaves. If we pick the best 100 leaves and trace them back through the tree, we expect to find that within a few times $\log n$ steps, they all converge to a small number of common ancestors.

Consequently, our proposed bubble decoder accepts two parameters: the *depth* d, and the *beam width* B. Instead of searching the entire decoding tree, we maintain B common ancestors, termed the *beam*, and a partial decoding tree rooted on each ancestor. The pseudo-code for the bubble decoder is:

```
Let T_0 be the tree of nodes out to depth d from root.
beam ← {T_0}    # set of rooted trees
for i = 1 to n/k − d do
    candidates ← [ ]    # list of (tree,cost) tuples
    for T ∈ beam do
        for T' ∈ subtrees(root(T)) do
            Expand T' from depth d − 1 to depth d.
            Compute and store path_cost in expanded nodes.
            cost ← min{path_cost(x) | x ∈ leaves(T')}
            candidates.append((T', cost))
    # get B lowest cost candidates, breaking ties arbitrarily
    beam ← best(candidates, B)
return best(candidates, 1)
```

These steps are depicted in Figure 3.

When $d = 1$, this method is the classical beam search (from AI), also termed the M-algorithm in communication systems [1]. When $d = n/k - \log_k B$, we recover the full ML decoder without any tree pruning.

4.4 Tail Symbols

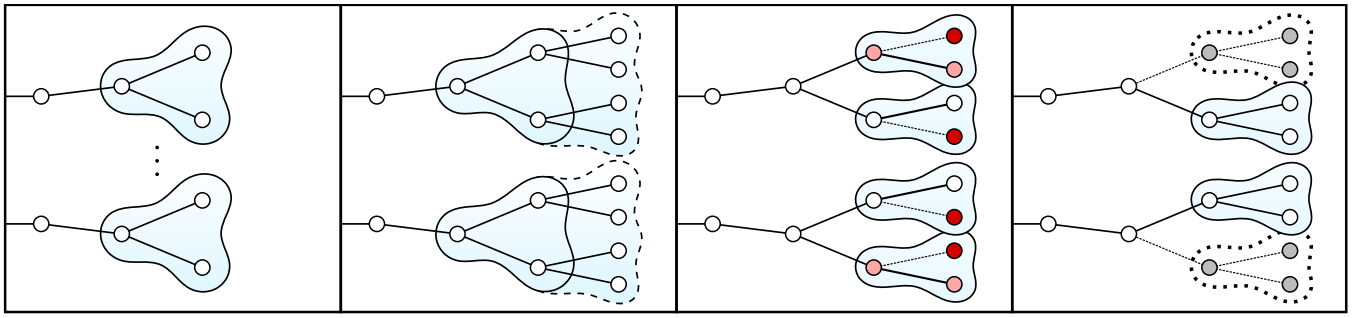
The bubble decoder ends up with a list of B messages, from which it should produce the best one. One approach is to reconstruct and validate all B messages, but the cost of doing so may be too high when B is large. An alternative is to send multiple symbols from $s_{n/k}$ in each pass (*tail symbols*), enough that if the correct candidate is in the beam, it has the lowest path cost. Then, only the lowest path cost message needs to be validated. We find that adding even just one tail symbol works well.

4.5 Decoding Time and Space Complexity

A single decoding attempt requires $n/k - d$ steps. Each step explores $B2^{kd}$ nodes at a cost of L RNG evaluations each, where L is the number of passes. Each step selects the best B candidates in $O(B2^k)$ comparisons using the selection algorithm. The overall cost is $O(\frac{n}{k} BL2^{kd})$ hashes and $O(\frac{n}{k} B2^k)$ comparisons.

Storage requirements are $O(B2^{kd}(k+v))$ for the beam and the partial trees, plus $O(\frac{n}{k} B(k + \log B))$ for message prefixes.

If B is polynomial in n, or if B is constant and $d = O(\log n)$, the total number of states maintained and the time complexity of operations remains polynomial in n. If both B and d are constant, the complexity of the bubble decoder is *linear* in n. Our experimental



(a) After step $i-1$ (b) Expand (c) Score (d) Pop

Figure 3: Sequential decoding process using the bubble decoder with $B = 2$, $d = 2$, $k = 1$. (a) At the beginning of step i, the partial trees have depth $d-1$. (b) Grow them out to depth d. (c) Propagate the smallest path costs back through the tree. (d) Select the B best children, pruning the rest. Advance to the next step and repeat.

results are for such linear-time configurations, with B maximized subject to a compute budget (§8.5).

In comparison, LDPC and Raptor decoders use several iterations of belief propagation (a global operation involving the entire message). Turbo decoders also require many full runs of the BCJR [2] or Viterbi algorithm [40]. All told, LDPC, Raptor, and turbo decoders perform several tens to thousands of operations per bit.

A spinal decoder with an appropriate choice of parameters performs a comparable number of operations per bit to these codes, achieves competitive throughput (§8), and is parallelizable (§7.2). The spinal decoder has the additional advantage that the decoder can run as symbols arrive because it operates sequentially over the received data, potentially reducing decode latency.

4.6 Capacity Results

For the AWGN channel with the uniform constellation mapping, we establish that a polynomial-time decoder achieves rates within a small constant (≈ 0.25 bits/symbol) of capacity. The proof appears in the appendix. A recent companion paper [3] states and establishes capacity results for the AWGN channel with the Gaussian constellation, and for the BSC: *the spinal decoder achieves capacity under these settings.*

THEOREM 1 (AWGN CHANNEL PERFORMANCE). *Let* $\mathsf{C}_{\mathsf{awgn}}(\mathsf{SNR})$ *be the AWGN channel capacity per channel use. With the uniform constellation, a bubble decoder polynomial in n achieves BER* $\to 0$ *as* $n \to \infty$ *for any number of passes L such that*

$$L\Big[\mathsf{C}_{\mathsf{awgn}}(\mathsf{SNR}) - \delta\Big] > k,$$

with the degree of the polynomial inversely proportional to $(\mathsf{C}_{\mathsf{awgn}}(\mathsf{SNR}) - \delta - k/L)$ *and*

$$\delta \equiv \delta(c, P^*, \mathsf{SNR}) \approx 3(1+\mathsf{SNR})2^{-c} + \frac{1}{2}\log\left(\frac{\pi e}{6}\right). \qquad (3)$$

This result suggests that with the uniform constellation mapping, by selecting a large enough $c = \Omega(\log(1+\mathsf{SNR}))$, it is possible to achieve rates within $\frac{1}{2}\log(\pi e/6) \approx 0.25$ of $\mathsf{C}_{\mathsf{awgn}}(\mathsf{SNR})$. As mentioned above, it is possible to close this theoretical gap with an appropriately chosen Gaussian constellation mapping. In simulation with finite n, however, we do not see significant performance differences between the Gaussian and uniform mappings.

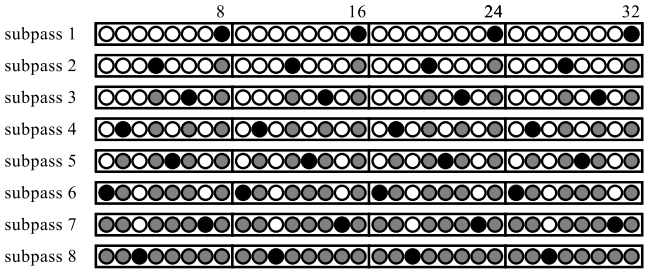


Figure 4: Puncturing schedule. In each subpass, the sender transmits symbols for spine values marked by dark circles; shaded circles represent spine values that have already been sent in a previous subpass.

5. PUNCTURING

In §3, the sender transmits one symbol per spine value per pass. If it takes ℓ passes to decode the message, the rate achieved is k/ℓ bits per symbol, with a maximum of k. Moreover, at moderate SNR, when ℓ is a small integer, quantization introduces plateaus in the throughput. Because the decoding cost is exponential in k, we cannot simply increase k to overcome these disadvantages.

Spinal codes may be *punctured* to achieve both high and finer-grained rates, without increasing the cost of decoding. Rather than sending one symbol per spine value per pass, the sender skips some spine values and, if required, fills them in subsequent "subpasses" before starting the next pass.

Figure 4 shows transmission schedule we implemented (others are possible). Each pass is divided into eight subpasses (rows in the figure). Within a subpass, only the spine values corresponding to dark circles are transmitted. Decoding may terminate after any subpass, producing a fine-grained set of achievable rates. This schedule nominally permits rates as high as $8k$ bits per symbol.

Puncturing does not change the decoder algorithm. For any missing spine value in a subpass, the associated branch costs are treated as 0, and the children are computed as before (all the children of a given parent will have the same score). If the correct candidate falls out of the beam, decoding will indeed fail in this subpass. If B is large enough, the correct candidate may remain in the beam until the next non-omitted spine value arrives. In our experiments, we find that $B = 256$ exhibits the positive benefits of puncturing; as computing becomes cheaper, increasing B further will cause the benefits of puncturing to be even more pronounced.

6. LINK-LAYER FRAMING

To use spinal codes, two changes to the traditional (e.g., 802.11) link-layer protocol are useful. First, because the code is rateless, the encoder and decoder must maintain some state across distinct frame transmissions, and use the *cumulative* knowledge of transmitted symbols to decode a message. The amount of data kept is small (on the order of a few kilobytes), similar to H-ARQ receivers implementing incremental redundancy (e.g., 3G, LTE). To prevent an erased frame transmission (e.g., the receiver fails to lock on to the preamble) from de-synchronizing the receiver (which needs to know which spine values are being sent in the frame), the sender should use a short sequence number protected with a highly redundant code (cf. the PLCP header in 802.11).

Second, it is useful to divide a single link-layer frame into multiple *code blocks*, each encoded separately. This use of code blocks is unusual, but not unique (cf. 802.11n with LDPC). The reason we use it is that, for a fixed compute budget at the decoder, shorter coded messages come closer to the Shannon capacity (§8). Each code block has a maximum length, n (1024 bits in our experiments).

At the link layer, the sender takes a datagram from the network layer and divides it into one or more code blocks of size not exceeding n bits. It computes and inserts a 16-bit CRC at the end of each block to construct a link-layer frame. This frame is handed to the encoder, which encodes each code block independently to produce symbols.

The sender transmits a certain number of symbols, and then pauses for feedback from the receiver. An important concern for any rateless code over half-duplex radios is that the receiver cannot send feedback when the sender is still transmitting, and the sender may not know when to pause for feedback. To achieve high throughput, a good algorithm is required for determining pause points. We have addressed this problem in more recent work [16].

At the receiver, the decoder processes the received symbols for each code block. If any block gets decoded successfully (the CRC passes), the next link-layer ACK indicates that fact. The ACK timing is similar to 802.11, but the ACK contains one bit per code block.

7. IMPLEMENTATION

This section describes implementation considerations for spinal codes. After describing general implementation principles, we describe the salient features of our hardware prototype.

The first goal when implementing a communication system is selecting the range of conditions under which we would like the system to perform well. For wireless networks, we expect the maximum SNR observed in practice to be 30 to 35 dB [14, 10]. At the lower end, we would like to support as low an SNR as possible, to work in challenging conditions with high external interference.

An implementation of spinal codes should pre-select (one or more) hash functions, RNGs, and values of k, perhaps even at the time of protocol standardization. Of course, these could be selected dynamically from a set of possibilities, but the sender and receiver need to agree on them. k determines the maximum possible rate (with our puncturing schedule, it is $8k$ bits/symbol), but the complexity of decoding is exponential in k, so smaller values have lower decoding cost. In §8.5, we find that $k = 4$ provides performance close to capacity for SNRs as high 35 dB.

An attractive property of spinal codes is that, given a value of k, the rate achieved under any given set of channel conditions depends *only* on the decoder's computational capabilities. The *same* encoded transmission can achieve a higher rate at a decoder that invests a greater amount of computation. With bubble decoding, each receiver can pick a B and d independently (so, for instance, a base station might pick values larger than a phone, and mobile devices could pick different values). The transmitter requires no knowledge about the receiver's capabilities, which avoids to need to negotiate supported modulation and coding schemes (bit rates) on every association.

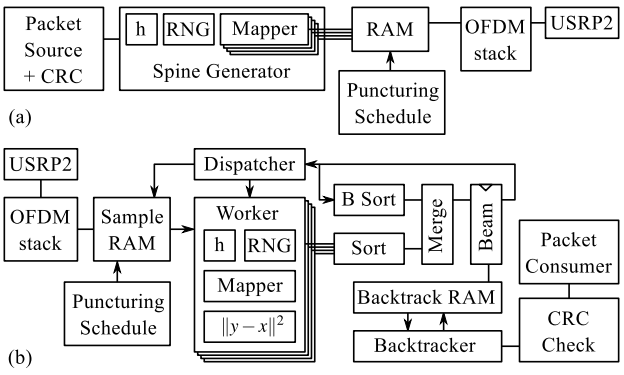


Figure 5: Hardware implementation block diagram, showing the spinal transmitter (a) and receiver (b).

7.1 Implementation Decisions

Choosing h. Spinal codes rely on the "mixing" ability of the hash function to provide pairwise independence. We initially used Salsa20 [5], a cryptographic-strength function with demonstrated mixing properties. On each use, Salsa20 requires 320 XORs, 320 additions and 320 rotations on 32-bit words. With these results in hand, we compared code performance with two other much cheaper hash functions developed by Jenkins, one-at-a-time and lookup3.[1] The one-at-a-time hash requires just 6 XORs, 15 bit shifts and 10 additions per application. Our simulations showed no discernible difference in performance between these three hash functions. We used one-at-a-time in our implementation and experiments.

RNG. We implemented RNG using one-at-a-time; to get the t^{th} output symbol, the encoder and decoder call $h(s_i,t)$. This method has the desirable property that not every output symbol has to be generated in sequence: if some frames containing symbols are not recovered, the decoder need not generate the missing symbols.

Other parameters. We find that $c = 6$, $B = 256$, $k = 4$, $d = 1$ are good choices of parameters; see §8.5 for supporting results.

PHY and link layers. The hardware implementation runs atop an OFDM PHY. It uses code block sizes of up to 1024 bits with a 16-bit CRC, dividing a longer packet into multiple 1024-bit code blocks.

Decoder details. The bubble decoder may be invoked multiple times on the same message with different numbers of input symbols. At first glance, it would seem like a good idea to cache explored nodes in the decoding tree between decoder runs, so in subsequent runs the scores would only need to be incrementally updated rather than recomputed. However, until enough symbols have arrived to successfully decode the message, the new symbols end up changing pruning choices to the extent that caching turns out to be unhelpful. Instead, the decoder stores the received symbols, and uses them to rebuild the tree in each run.

7.2 Hardware Implementation

For high-speed, low-power wireless operation, a code must be feasible in hardware. To demonstrate feasibility, we implemented a prototype spinal encoder and $d = 1$ bubble decoder using the

[1] `http://en.wikipedia.org/wiki/Jenkins_hash_function`

Airblue [25] platform. We incorporated spinal codes into Airblue's 802.11 OFDM stack to create 802.11-like 20 MHz and 10 MHz OFDM transceivers (Figure 5).

Spinal codes are attractive to implement in hardware because of the high parallelism and low latency made possible by their tree-like structure. These properties contrast with existing high-performance codes like turbo and LDPC, which have limited parallelism and longer latency due to their iterative structure. Although the spinal encoder is a straightforward sequence of hashes, RNG evaluations, and constellation mappings, the decoder requires careful design to take advantage of parallelism.

As samples arrive from the OFDM stack, they are written into an SRAM in unpunctured order, with passes for a given spine value located at adjacent addresses for batch reads. When a decode attempt starts, a dispatch unit instructs M identical worker units to explore all possible decodings of the first k bits, starting from state s_0. Each worker has a certain number of hash units, which serve double duty for computing h and RNG. A worker explores a node by computing several hashes per cycle until it has mapped, subtracted, squared, and accumulated the branch cost over all available passes.

Over the course of several cycles, the dispatcher and the workers will deliver $B2^k$ scored candidate nodes to the selection unit. This stage, corresponding to the two inner loops in the algorithm in §4.3, is highly parallelizable: the work accomplished per cycle is linear in the number of workers.

These candidates stream into a selection unit, which identifies the best B of them. The selection unit sorts the M candidates delivered in a given cycle, selecting the best B. The candidates from prior cycles will have already been winnowed down to the best B, so the system merges those with the B from this cycle. The result is B items in bitonic (not sorted) order. The system stores this list in a register, and on the next cycle finishes sorting these B in parallel with sorting the new M.

Once all $B2^k$ nodes have been scored and selected, the best B become the new beam, and are copied to the backtrack memory. This step advances the outer loop of the algorithm. On the last iteration, the system fully sorts the B candidates and picks the best one, then follows backtrack pointers to recover the message, and checks its CRC.

This prototype has a throughput of up to 10 Mbps in FPGA technology. Synthesized using the Synopsis Design Compiler for the TSMC 65 nm process, the design can sustain 50 Mbps. This decoder is competitive with algorithms like Viterbi decoding in terms of logic area (.60 mm^2 versus .12 mm^2), which is encouraging considering the decades of research and development devoted to Viterbi decoding. In more recent work, we have developed a hardware spinal decoder (with some refinements and generalizations to the above approach) that is competitive with turbo decoding.

8. EVALUATION

Our goal is to evaluate spinal codes under various conditions and compare it to the following codes:

LDPC. We use the same combinations of code rates and modulations for our LDPC implementation as in 802.11n [19], using soft demapped information. The code block size $n = 648$ bits. We implemented a belief propagation decoder that uses forty full iterations with a floating point representation [39]. To mimic a good bit rate adaptation strategy such as SoftRate [41] working atop the LDPC codes, we plot the *best envelope* of LDPC codes in our results; i.e., for each SNR, we report the highest rate achieved by the entire family of LDPC codes.

Raptor code. We follow a similar construction optimized for the AWGN channel to Yedidia & Palanki [26], with an inner LT code generated using the degree distribution in the Raptor RFC [22], and an outer LDPC code as suggested by Shokrollahi [33] with a forty-iteration belief propagation decoder. The outer code rate is 0.95 with a regular left degree of 4 and a binomial right degree. We experimented with different symbol sets, and report results for the dense QAM-256 constellation as well as QAM-64. We calculate the soft information between each received symbol and the other symbols, a process that takes time exponential in the number of constellation points: QAM-2^α requires time $\Theta(2^{\alpha/2})$.

Strider. Our Strider implementation is a C++ port of the Matlab software from Gudipati [12]. We use the recommended 33 data blocks (layers), a rate-$1/5$ base turbo code with QPSK modulation, and up to 27 passes. Unless mentioned otherwise, we use the recommended code block size of 50490 bits. A significant enhancement we added to Strider is *puncturing*, to enable it to achieve a finer-grained set of rates than in the original work (denoted by "Strider+").

8.1 Experimental Setup and Metrics

Software platform. To evaluate the different codes under the same conditions, we integrated all codes into a single framework, built with no sharing of information between the transmitter and receiver components. A generic *rateless execution engine* regulates the streaming of symbols across processing elements from the encoder, through the mapper, channel simulator, and demapper, to the decoder, and collects performance statistics. All codes run through the same engine. In most cases, we measure performance across an SNR range from -5 dB to 35 dB, stepping by 1 dB at a time.

Hardware experiments. We use on-air hardware experiments to cross-validate the software platform results and to demonstrate that spinal codes perform well in hardware under real-world, wideband conditions. We use high-speed transceivers constructed using Airblue [25], which is built out of Xilinx XUPV5 FPGA boards and USRP2 radio front-ends. All on-air experiments were conducted in the 2.4 GHz ISM band in an unshielded laboratory at MIT CSAIL. We tested spinal codes with both 20 MHz and 10 MHz waveforms.

Metrics. We evaluate two metrics: the *rate* and the *gap to capacity*. We measure the rate in bits per symbol, so multiplying that number by the channel bandwidth (in Hz), and subtracting OFDM overheads, would give the throughput in bits per second.

The gap to capacity is often a more instructive metric than the rate because it allows us to compare how close different codes are to the Shannon limit. The "gap to capacity" of a code, $\mathscr{C}$, at a given SNR, is defined as how much more noise a capacity-achieving code can handle and still provide the same throughput as $\mathscr{C}$. For example, say a code achieves a rate of 3 bits/symbol at an SNR of 12 dB. Because the Shannon capacity is 3 bits/symbol at 8.45 dB, the gap to capacity is $8.45 - 12 = -3.55$ dB.

8.2 AWGN Channel Performance

Figure 6 shows three charts comparing Raptor codes, Strider, and LDPC codes to spinal codes from experiments run on the standard code parameters for each code. The first two charts show the rates as a function of SNR, while the third shows the gap to capacity. The two spinal code curves (256 and 1024 bits) both come closer to Shannon capacity than any of the other codes across all SNR values from -5 dB to 35 dB. The gap-to-capacity curves show that spinal codes consistently maintain a smaller gap than all the other codes.

We aggregate by SNR to summarize gains under different conditions. Above an SNR of 20 dB, spinal codes obtain a rate 21% higher than Raptor/QAM-256, 40% higher than Strider, and 54% higher than the LDPC envelope. Between 10 and 20 dB, spinal codes achieve a rate 25% higher than Strider and 12% higher than Raptor/QAM-256. At SNRs below 10 dB, spinal codes achieve a rate 20% higher than Raptor/QAM-256 and 32% higher than Strider.

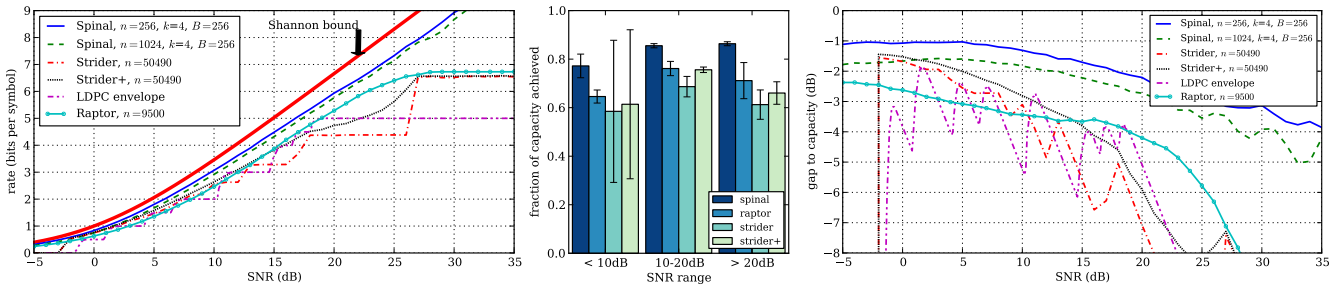


Figure 6: Rates achieved by spinal code with $k = 4$, $B = 256$, $d = 1$, and the other codes (Strider+ is Strider with our puncturing enhancement). Experiments at each SNR average Raptor performance over 100-300 kbits of data, Strider over 5-20 Mbits, LDPC over 2 Mbits, and spinal codes over 0.6 to 3 Mbits.

Strider. Strider uses 33 parallel rate-$1/5$ turbo codes with QPSK modulation, so without puncturing, the rates it achieves track the expression $(2/5) \cdot 33/\ell$ bits/symbol, where ℓ is the number of passes required for successful decoding. In the tested SNR range, Strider needs at least $\ell = 2$ passes to decode, for a maximum rate of 6.6 bits/symbol. The puncturing enhancement we added (Strider+) produces the more graded set of achieved rates shown in Figure 6. At low SNR, we find that Strider is unable to successfully decode as many messages as spinal codes. Another source of inefficiency in Strider is that the underlying rate-$1/5$ turbo code has a non-negligible gap to capacity. The results (without puncturing) are generally consistent with Figure 4a in the Strider paper [12]; it is important to note that the "omniscient" scheme discussed in that paper is constrained to modulation and coding schemes in 802.11a/g, and as such has a significant gap to the Shannon capacity.

Raptor. We are unaware of any previously reported Raptor result for the AWGN channel that achieves rates as high as those shown in our implementation [26]. We believe that one reason for the good performance is that we have a careful demapping scheme that attempts to preserve as much soft information as possible. That said, spinal codes still perform 12%–21% better across the entire SNR range, with the greatest gains at low and high SNRs. There are two reasons for better performance: first, spinal codes naturally incorporate soft information, while Raptor (and also Strider) loses information in the mapping/demapping steps, and second, the LT code used in Raptor has some information loss. We experimented with Raptor/QAM-64 as well, finding that it performs a little better at low-to-medium SNR (16% worse than spinal codes, rather than 20%), but does much worse (54%) at high SNR. The dense QAM-256 constellation does entail a significantly higher decoding cost for Raptor, whereas spinal codes naturally support dense constellations.

LDPC. The primary reason why spinal codes do better than the best envelope of LDPC codes has to do with the ability of rateless codes to take advantage of "lucky" channel conditions. We term this the *hedging effect*. Intuitively, hedging is the ability to decode in less time when the noise is low, without sacrificing reliability. This property is more general than the LDPC comparison. In particular, Figure 7 demonstrates that the *rateless* spinal code outperforms every *rated* version of the spinal code at every SNR.

Constant SNR means that the distribution of the noise does not vary, but the realized noise does vary substantially over time. Because rated codes cannot adapt to realized noise, they must be risk-averse to ensure a high probability of decoding. Hence, they tend to occupy the channel for longer than strictly necessary. By contrast, rateless codes can use the channel for less time when the realized noise is small and thus achieve higher rates. Due to the law of large

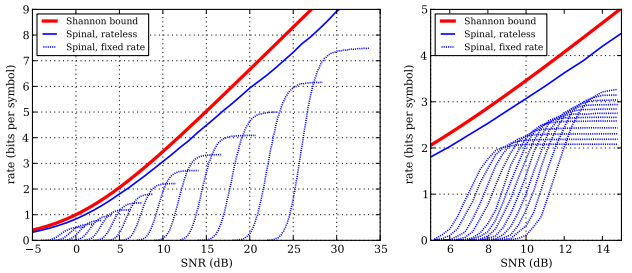


Figure 7: Throughput of the rateless spinal code compared to various rated versions of the spinal code.

numbers (precisely, concentration), this effect diminishes with increasing message length. For the same reason, rated codes approach capacity only for long message sizes.

Small code block sizes. The results presented above picked favorable code block (message) sizes for each code. For many Internet applications, including audio and games, the natural packet size is in the 64-256-byte range, rather than tens of thousands of bits. Understanding the performance of different codes in this regime would help us evaluate their effectiveness for such applications.

Figure 8 shows the rates achieved by spinal codes, Raptor, and Strider at three small packet sizes: 1024, 2048, and 3072 bits. Each column shows the results obtained for data transfers in the SNR range 5 to 25 dB. In this range, spinal codes outperform Raptor by between 14% and 20% for these packet sizes.

The gains over Strider are substantial (2.5× to 10×) even when puncturing is used. To handle small packets in Strider, we used the same number of layers and reduced the number of symbols per layer, which is a reasonable method. It is possible that reducing the number of layers might help, but it is unclear how best to handle smaller sizes in Strider.

8.3 Hardware Over-the-air Results

Figure 9 shows the results obtained by measuring the Airblue spinal code implementation in over-the-air experiments for $n = 192$ bits, $k = 4$, $c = 7$, $d = 1$, and $B = 4$. Each $+$ sign in the figure is the rate measured by transmitting at least 20 messages over a 10 MHz band. The measured on-air decoding performance closely tracks the results of a similarly configured software simulator across a large SNR range (the range achievable using commodity USRP2/RFX2400 radio frontends), providing important real-world validation of the code's performance. Differences include effects of fixed-point precision, but should not affect the take-away point: a reasonable implementation is both achievable and operational.

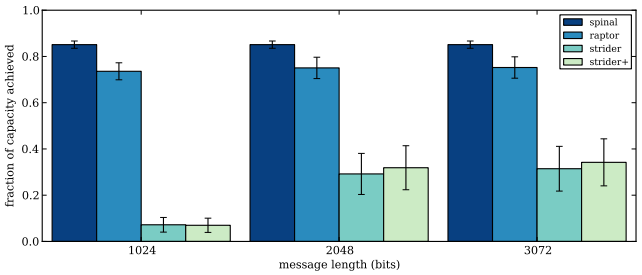


Figure 8: Average fraction of capacity in range 5-20 dB for spinal codes, Raptor and Strider at different message sizes.

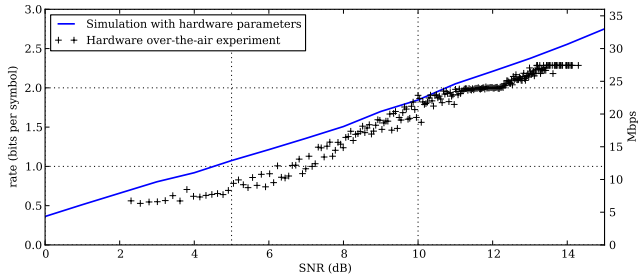


Figure 9: Rates achieved by the hardware over-the-air experiment, compared to a software simulation with similar parameters. Throughput (right axis) shows equivalent link rate for a 20 MHz 802.11a/g channel.

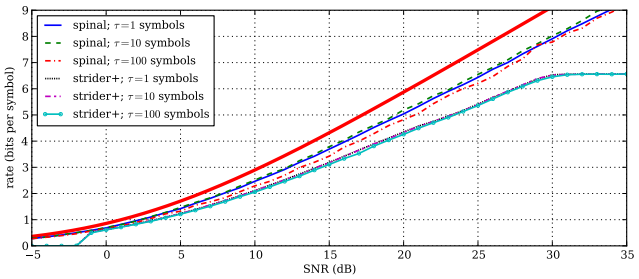


Figure 10: Performance of spinal codes and strider in a simulation model of a Rayleigh fading environment. The decoders are given exact fading channel parameters.

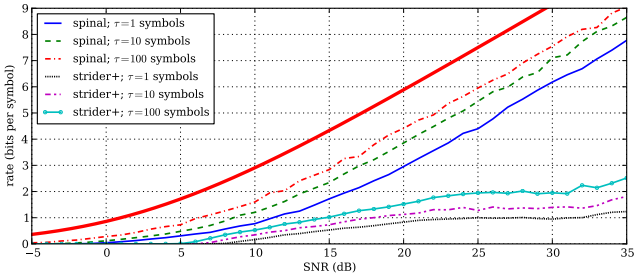


Figure 11: Performance of the AWGN decoders on the Rayleigh simulation. This experiment examines the decoders' resilience to varying or inaccurate channel information.

8.4 Fading Channel Performance

This section describes experiments with spinal codes and Strider (with our puncturing enhancement) over a fading channel model [36]. The model is a Rayleigh fading environment with two parameters (σ^2, τ). The transmitted signal x is transformed according to $y = hx + n$, where y is the received signal, n is Gaussian noise of power σ^2, and h is a complex fading coefficient randomized every τ symbols to a complex number with uniform phase and Rayleigh magnitude.

The first experiment shows the performance of the codes on fading channels, with both codes incorporating detailed fading information. In the second experiment, neither decoder is given fading information. As such, this experiment evaluates the robustness of the codes to varying conditions and to inaccurate estimates of channel parameters, as might occur in practice.

Figures 10 and 11 show the results of both experiments for three different coherence times, specified as multiples of one symbol time. In both graphs, the top curve is the capacity of the *fading* channel. It is noteworthy that spinal codes perform roughly similarly at all the measured coherence times when fading information is available. Compared to Strider+, at 10 dB, the improvement is between 11% and 20% (without puncturing in Strider, the gains are between 19% and 28%). At an SNR of 20 dB, the gains are between 13% and 20% (without puncturing, between 28% and 33%). When no fading information is available, spinal codes achieve much higher rates than Strider+ (Figure 11).

These results show that spinal codes perform well across a wide range of time-varying conditions, and that spinal decoding is robust even when the decoder does not have accurate fading information.

8.5 Exploration of Spinal Code Parameters

Collision probability. Spines for two distinct messages can converge when there is a hash collision, i.e., $h(s_i, \bar{m}_i) = h(s'_i, \bar{m}'_i)$. Collisions degrade the decoder's ability to discriminate between candidate messages with different prefixes. The probability that colliding messages exist can be made exponentially small in the message length n by choosing $\nu \geq 3n$ (cf. the Birthday Paradox).

In practice, it is not necessary to eliminate all collisions to achieve high performance. A collision potentially reduces performance if it occurs between the correct message and another candidate in the beam. Each iteration explores $B2^{kd}$ nodes. In a decode attempt, a node collides with the correct one with probability $\sim (n/k)2^{-\nu}B2^{kd}$, so these events are rare if $\nu \gg \log(B) + \log(n) + kd$. For example, with $n = 256$, $k = 4$, $B = 256$, $d = 1$, and $\nu = 32$, a collision occurs only once per 2^{14} decodes on average.

Picking k and B. Figure 12 shows that $k = 4$ performs well across a range of compute budgets (the x axis is proportional to $B2^k/k$). Smaller values of k under-perform at higher SNRs; larger values of k don't do well at low compute budgets. Each decoder can use a value of B according to its price/performance ratio. As computation becomes cheaper, increasingly higher budgets can be used, translating to higher B, to get better performance. From this graph, we conclude that $k = 4$ is a good choice for the SNR range we are targeting. For our experimental compute budgets, $B = 256$ is a reasonable choice.

Picking c. The number of output bits, c, limits the maximum throughput. When c is small, even if the channel's SNR can support a high rate, there are simply too few bits transmitted to decode with high throughput. Figure 13 shows that $c = 6$ is the right choice for the range of SNR values we are concerned with.

Peak-to-average power ratio (PAPR). A practical modulation scheme should have a modest PAPR, defined in terms of the output waveform $y(t)$ as $10 \cdot \log_{10} \frac{\max |y(t)|^2}{\text{mean}|y(t)|^2}$. High PAPR is a problem because the linearity of radio components degrades when waveforms have large peaks. In a non-OFDM wireless system, dense constel-

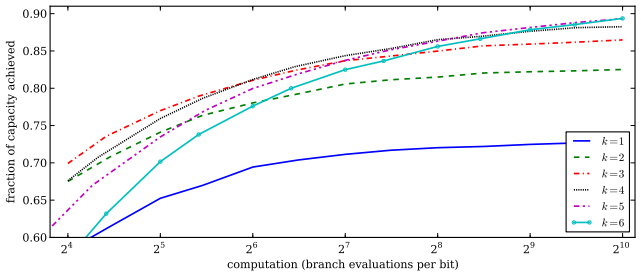


Figure 12: How compute budget per bit ($B2^k/k$) affects performance in the SNR range 2-24 dB, for different k. A choice of $k = 4$ yields codes that perform well over the entire range of budgets. This graph also shows that $B = 256$ is a good choice.

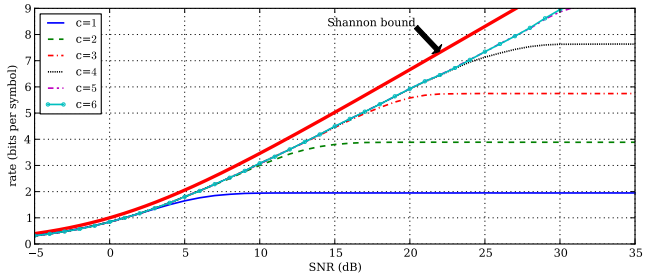


Figure 13: Throughput with different densities of output symbols. $c = 6$ is a good choice for this range of SNRs.

lations usually have a high PAPR: for QAM-4 it is 0 dB, while for QAM-∞ it is 4.77 dB.

These results, however, do not carry over to the 802.11a/g OFDM stack, which our PHY uses. For such OFDM systems using scrambling, PAPR is typically 5-12 dB [29], depending on the transmitted symbols. As shown in Table 1, OFDM obscures all but negligible differences between the PAPRs of dense constellations and standard WiFi constellations.

Constellation	Mean PAPR	99.99% below
QAM-4	7.34 dB	11.31 dB
QAM-64	7.31 dB	11.41 dB
QAM-2^{20}	7.31 dB	11.43 dB
Trunc. Gaussian, $\beta = 2$	7.29 dB	11.47 dB

Table 1: Empirical PAPR for 802.11a/g OFDM with various constellations, showing negligible effect of constellation density. Each row summarizes 5 million experiments.

Code block length. A strength of the spinal code is good memory in the encoding, so bad information from a burst of noise can be corrected by all following symbols if necessary. But this memory also has a price: once a path is pruned out, the probability of the decoder resynchronizing to a useful path is low. The decoder has to receive more symbols until the true path is not pruned. However small this probability, for fixed k and B, a longer code block means more opportunities for the true path to be lost. Hence, longer code blocks require either more symbols per bit or a larger B in order to decode, even with the same SNR, as reflected in Figure 14.

9. Conclusion

This paper described the design, implementation, and evaluation of rateless spinal codes. The key idea in spinal codes is the sequential

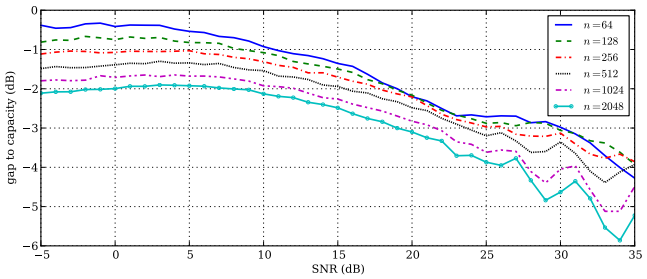


Figure 14: Effect of code block length on performance ($k = 4$, $B = 256$). Some puncturing artifacts can be seen above 25 dB, where less than one pass is transmitted on average.

application of a random hash function to the message bits to produce a sequence of coded bits and symbols for transmission. We described a novel, efficient, capacity-achieving bubble decoder for spinal codes. Our experimental results show that spinal codes out-perform Raptor, Strider, and the best envelope of 802.11n LDPC codes by significant amounts over a range of channel conditions and code block sizes. Our hardware prototype in Airblue [25] runs at 10 Mbits/s on FPGA hardware, and we estimate it can run at 50 Mbits/s in silicon.

This paper opens up several avenues for future work. First, developing a wireless network architecture atop spinal codes that provides a different wireless link abstraction from today: a link is that is always reliable at all SNR above some well-defined threshold, but which produces outages below the threshold, eliminating highly variable packet delays. Second, developing a good link-layer protocol for rateless codes to deal with the issues raised in §6. Third, developing a software "shim" layer using spinal codes layered over a bit-error channel such as an existing wireless link that uses a sub-optimal coding/modulation scheme. Fourth, investigating the joint-decoding properties of codes that use hash functions. And last but not least, the ideas presented in this paper may provide a constructive framework for de-randomizing, and realizing in practice, a variety of random-coding arguments widely used in information-theoretic proofs.

Acknowledgments

We thank Joseph Lynch for helping us collect hardware results and Aditya Gudipati for support in implementing Strider. We thank David Andersen, Nick Feamster, Daniel Halperin, Mark Handley, Kyle Jamieson, Henry Pfister, Tom Richardson, Pramod Viswanath, Lizhong Zheng, and the SIGCOMM reviewers for helpful comments. Three generous graduate fellowships supported this work: the Irwin and Joan Jacobs Presidential Fellowship (Perry and Iannucci); the Claude E. Shannon Assistantship (Perry), and the Intel Fellowship (Fleming). Intel also partially supported the Airblue platform. 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] J. Anderson and S. Mohan. Sequential coding algorithms: A survey and cost analysis. *IEEE Trans. on Comm.*, 32(2):169–176, 1984.

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

[3] H. Balakrishnan, P. Iannucci, J. Perry, and D. Shah. De-randomizing Shannon: The Design and Analysis of a Capacity-Achieving Rateless Code. *arXiv:1206.0418*, June 2012.

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

[5] D. Bernstein. The Salsa20 Family of Stream Ciphers. *Lecture Notes in Computer Science*, 4986:84–97, 2008.

[6] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes (1). In *ICC*, 1993.

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

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

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

[10] J. Frigon and B. Daneshrad. Field measurements of an indoor high-speed QAM wireless system using decision feedback equalization and smart antenna array. *IEEE Trans. Wireless Comm.*, 1(1):134–144, 2002.

[11] R. Gallager. Low-density parity-check codes. *IRE Trans. Information Theory*, 8(1):21–28, 1962.

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

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

[14] D. Halperin, W. Hu, A. Sheth, and D. Wetherall. Predictable 802.11 packet delivery from wireless channel measurements. In *SIGCOMM*, 2010.

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

[16] P. Iannucci, J. Perry, H. Balakrishnan, and D. Shah. No Symbol Left Behind: A Link-Layer Protocol for Rateless Codes. In *MobiCom*, 2012.

[17] F. Jelinek. Fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, 13(6):675–685, 1969.

[18] G. Judd, X. Wang, and P. Steenkiste. Efficient Channel-aware Rate Adaptation in Dynamic Environments. In *MobiSys*, June 2008.

[19] IEEE Std 802.11n-2009: Enhancements for Higher Throughput.

[20] 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.

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

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

[23] R. Mantha and F. Kschischang. A capacity-approaching hybrid ARQ scheme using turbo codes. In *GLOBECOM*, 1999.

[24] M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*, chapter 13, pages 321–326. Cambridge University Press, 2005.

[25] M. C. Ng, K. E. Fleming, M. Vutukuru, S. Gross, Arvind, and H. Balakrishnan. Airblue: A System for Cross-Layer Wireless Protocol Development. In *ANCS*, Oct. 2010.

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

[27] J. Perry, H. Balakrishnan, and D. Shah. Rateless Spinal Codes. In *HotNets-X*, Oct. 2011.

[28] G. Pottie and D. Taylor. A comparison of reduced complexity decoding algorithms for trellis codes. *JSAC*, 7(9):1369–1380, 1989.

[29] C. Schurgers and M. B. Srivastava. A Systematic Approach to Peak-to-Average Power Ratio in OFDM. In *SPIE's 47th Meeting*, 2001.

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

[31] S. Sesia, G. Caire, and G. Vivier. Incremental redundancy hybrid ARQ schemes based on low-density parity-check codes. *IEEE Trans. on Comm.*, 52(8):1311–1321, 2004.

[32] C. Shannon. Communication in the presence of noise. *Proc. of the IRE*, 37(1):10–21, 1949.

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

[34] N. Shulman. *Universal channel coding*. PhD thesis, Tel-Aviv University, 2004.

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

[36] E. Telatar. Capacity of Multi-Antenna Gaussian Channels. *European Trans. on Telecom.*, 10(6):585–595, 1999.

[37] G. Ungerboeck. Channel coding with multilevel/phase signals. *IEEE Trans. Info. Theory*, IT-28(1):55–67, Jan. 1982.

[38] G. Ungerboeck and I. Csajka. On improving data-link performance by increasing the channel alphabet and introducing sequence coding. In *ISIT*, 1976.

[39] A. Vila Casado, M. Griot, and R. Wesel. Informed dynamic scheduling for Belief-Propagation decoding of LDPC codes. In *IEEE ICC*, 2007.

[40] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Info. Theory*, 13(2):260–269, 1967.

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

[42] S. Wicker and V. Bhargava. *Reed-Solomon codes and their applications*. Wiley-IEEE Press, 1999.

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

APPENDIX: PROOF SKETCH OF THEOREM 1

We establish the theorem for the real-valued AWGN channel; the complex channel simply doubles the capacity. We only cover the uniform constellation mapping here. This argument applies to a bubble decoder with depth $d = 1$ and width B polynomial in n (exponent to be determined). The proof for $d' \neq 1$, $B' \geq \max(1, 2^{-kd}B)$ follows similarly and is omitted for space.

Inputs to the constellation mapping function are uniformly distributed, thanks to the mixing properties of the hash function and RNG. Under the uniform mapping, these inputs are mapped directly to the range $[-\sqrt{3P/2}, \sqrt{3P/2}]$, giving an average symbol power (variance) slightly less than $P^* \triangleq P/2$; the difference vanishes as $c \to \infty$. The Shannon capacity [32] of the AWGN channel with average power constraint P^* is

$$\mathsf{C}_{\mathsf{awgn}}(P^*) = {}^1\!/_2 \log_2(1+\mathsf{SNR}) \text{ bits/symbol}, \tag{4}$$

where $\mathsf{SNR} = \frac{P^*}{\sigma^2}$ denotes the signal-to-noise ratio.

The rate of the spinal code after L passes is k/L. Reliable decoding is not possible until L passes have been received such that k/L is less than $\mathsf{C}_{\mathsf{awgn}}(P^*)$. We shall show that for essentially the smallest L that satisfies this inequality, our polynomial-time decoder will produce the correct message with high probability. In the remainder of this section, we assume L is the smallest value such that $k/L < \mathsf{C}_{\mathsf{awgn}}(P^*) - \delta(c, \mathsf{SNR}, P^*)$. Without puncturing, a single-pass transmission of the code results in rate $R_{max} = n/(n/k) = k$. That is, the rate after L passes, k/L, is R_{max}/L.

The spinal encoder (§3) generates $x_{i,\ell}$, the i^{th} symbol of the ℓ^{th} pass, from the spine value s_i, $1 \leq i \leq n/k$, $1 \leq \ell \leq L$. Now s_i depends on s_{i-1} and the k bits $\bar{m}_i$, s_{i-1} in turn depends on s_{i-2} and $\bar{m}_{i-1}$, and so on. Choosing v large enough to avoid collisions with extremely high probability, and using the pairwise independence properties of h and RNG, $x_{i,\ell} = f(h_{i,\ell}(s_0, \bar{m}_1, \ldots, \bar{m}_i))$, where $h_{i,\ell}$ is a random hash function satisfying pairwise independence, and f is the deterministic constellation mapping. That is, the first i symbols of any pass ℓ depend only on the first ik bits, denoted by $M_i = (m_1, \ldots, m_{ik})$. Therefore, two codewords that differ only in the bits indexed higher than ik get mapped to the same first i symbols (in each pass), but all subsequent symbols generated for them are independent of each other and entirely random.

After receiving symbols for L passes, the bubble decoder sequentially expands the decoding tree, maintaining up to B candidate message prefixes $M_i^1, \ldots, M_i^B$ at each stage i with the hope that for $i = n/k$, they will be closest messages $M^1, \ldots, M^B \in \{0,1\}^n$ to the received $\bar{y}$: i.e., that $\bar{x}(M^1), \ldots, \bar{x}(M^B)$ minimize $\|\bar{y} - \bar{x}(M')\|$ over all $M' \in \{0,1\}^n$.

For convenience, the theorem modifies the bubble decoder slightly, replacing $\bar{y}$ with $\alpha\bar{y}$ where $\alpha = P^*/(P^* + \sigma^2)$. (This modification effectively is a linear MMSE decoder rather than ML, and is weaker than ML.) A capacity proof for this modified weaker bubble decoder implies that the result holds for the original.

Our goal is to show that, with high probability, at every stage of decoding, the bubble decoder has the prefix M_i of the correct

message M in its beam of B candidates. We shall additionally show that in this situation, the B candidate messages differ from each other (and hence from the correct message) only in the last $O(\log n)$ bits. If the decoder declares any one of the B candidates as the decoded message, the bit error rate (BER) is $O(\frac{\log n}{n}) \to 0$. That is, the BER goes to 0 as desired.

Intuitively, good messages have low path costs and bad messages have high path costs. Provided we are δ away from the Shannon bound, the difference between good and bad is large enough to make a reliable hard-decoding decision. We establish two invariants, which together give the desired result. The first is an upper bound on the path cost of the correct message, independent of constellation map. The second is a lower bound on the path cost of an incorrect message, and depends on the constellation map.

Invariant 1. Given message $M \in \{0,1\}^n$, the encoder transmits symbols $\bar{x}(M)$. Let $\bar{x}_i(M) = (x_{i,1}(M), \ldots, x_{i,L}(M))$ denote the first L symbols generated from the i^{th} spine value of M and let $\bar{y}_i$ be their noisy versions at the receiver. Then, for small enough $\varepsilon > 0$, with probability $1 - O(\exp(-\Theta(\varepsilon^2 iL)))$,

$$\sum_{j=1}^{i} \sum_{\ell=1}^{L} \big(\alpha y_{j,\ell} - x_{j,\ell}(M)\big)^2 \le (1+\varepsilon)\frac{iLP^*}{1+\mathsf{SNR}}, \tag{5}$$

for all $1 \le i \le L$. To see why, consider the following: for each j, ℓ, under the AWGN channel,

$$y_{j,\ell} = x_{j,\ell}(M) + z_{j,\ell}, \tag{6}$$

where $z_{j,\ell}$ is independent Gaussian noise (mean 0, variance σ^2). Therefore,

$$\begin{aligned}\Big(\alpha y_{j,\ell} - x_{j,\ell}(M)\Big)^2 &= \alpha^2 z_{j,\ell}^2 + (1-\alpha)^2 x_{j,\ell}(M)^2 \\ &\quad - 2\alpha(1-\alpha) z_{j,\ell} x_{j,\ell}(M).\end{aligned} \tag{7}$$

By the independence of $x_{j,\ell}(M)$ and $z_{j,\ell}$, and because $\mathbb{E}[x_{j,\ell}(M)^2] \approx P^*$, the mean of the RHS of (7) is

$$\begin{aligned}\alpha^2\sigma^2 + (1-\alpha)^2 P^* &= \frac{(P^*)^2\sigma^2}{(P^*+\sigma^2)^2} + \frac{P^*\sigma^4}{(P^*+\sigma^2)^2} \\ &= \frac{P^*\sigma^2}{(P^*+\sigma^2)} = \frac{P^*}{1+\mathsf{SNR}}.\end{aligned} \tag{8}$$

Because all the summands on the LHS of (5) are independent and identically distributed (i.i.d.), and there are iL summands in total, the mean of the LHS is precisely $iLP^*/(1+\mathsf{SNR})$. Now the LHS of (5) can be written as three summations, each having iL terms, one each corresponding to the terms on the RHS of (7). Because each of these is a summation of i.i.d. random variables with exponentially decaying tails (and $x_{j,\ell}^2$ is bounded by $3P/2$), applying a Chernoff-style bound implies a concentration of these terms around their means within error $\varepsilon iLP^*/(1+\mathsf{SNR})$), with probability decaying as $O\big(\exp\big(-\Theta(\varepsilon^2 iL)\big)\big)$ for small enough $\varepsilon > 0$. This argument completes the justification of (5).

For $iL = \Omega(\varepsilon^{-2}\log n)$, the bound holds with probability at least $1 - 1/n^4$. Hence, by the union bound ("the probability that at least one of the events happens is no greater than the sum of the probabilities of the individual events"), it holds simultaneously for all $i = \Omega(\varepsilon^{-2}\log n)$ with probability $\ge 1 - O(1/n^3)$. We need this bound for any contiguous set of indices $(q, q+1, \ldots, q+i)$ with $i = \Omega(\varepsilon^{-2}\log n)$. Since there are $O(n)$ such intervals, by another application of the union bound, this claim holds true with probability at least $1 - O(1/n^2)$.

Invariant 2. Consider $M' = (m'_1, \ldots, m'_n)$ with $m'_1 \ne m_1$, i.e., M' and $M = (m_1, \ldots, m_n)$ differ at least in the first bit. All the coded symbols of M and M' are mapped independently and at random. That is, $\bar{x}(M')$ is independent of $\bar{y}$. We'll focus, for the time being, on the first iL symbols. Now for the uniform constellation, one way to obtain $\bar{x}(M')$ is to sample a point uniformly at random in the cube $[-\sqrt{3P/2}, \sqrt{3P/2}]^{iL}$, and then map the co-ordinates in each of the iL dimensions to the nearest quantized value (at granularity $2^{-c}\sqrt{6P}$). Therefore, the probability of $\bar{x}(M')$ falling within squared distance $(1+\varepsilon)iLP^*/(1+\mathsf{SNR})$ of $\alpha\bar{y}$ is bounded above by the probability that a uniformly sampled point in $[-\sqrt{3P/2}, \sqrt{3P/2}]^{iL}$ falls within $r^2 \equiv (1+\varepsilon+\delta_1)iLP^*/(1+\mathsf{SNR})$ of $\alpha\bar{y}$, with $\delta_1 = 6(1+\mathsf{SNR})2^{-c}$. For the uniform distribution, this is merely the ratio of the volume of a ball of radius r and the volume of the cube $[-\sqrt{3P/2}, \sqrt{3P/2}]^{iL}$: using Stirling's approximation that $\ln K! \sim K\ln K - K$,

$$\begin{aligned}\frac{1}{(iL/2)!}\Big(\frac{\pi r^2}{6P}\Big)^{iL/2} &= \frac{1}{(iL/2)!}\Big(\frac{\pi(1+\varepsilon+\delta_1)iL}{12(1+\mathsf{SNR})}\Big)^{iL/2} \\ &\approx \Big(\frac{\pi e(1+\varepsilon+\delta_1)}{6(1+\mathsf{SNR})}\Big)^{iL/2} \\ &\approx 2^{-iL(\mathsf{C}_{\mathsf{awgn}}(P^*)-\Delta)},\end{aligned} \tag{9}$$

where $\Delta \approx \frac{1}{2}\big(\varepsilon + \delta_1 + \log(\pi e/6)\big)$.

Completing the proof using Invariants 1 and 2. Consider a bubble decoder at stage i trying to estimate M_i using $\bar{y}^i$. From Invariants 1 and 2, conditional on event (5) happening (which happens with high probability for $i = \Omega(\varepsilon^{-2}\log n)$), the chances of an M', differing from M in the first bit, having $\bar{x}^i(M')$ closer to $\alpha\bar{y}^i$ compared to $\bar{x}^i(M)$, is given by (9). There are at most 2^{ik-1} such messages. By the union bound, the chance for such an event is at most

$$P(i) = 2^{-iL\left(\mathsf{C}_{\mathsf{awgn}}(P^*)-\Delta-\frac{k}{L}\right)}. \tag{10}$$

Thus, if $k/L < \mathsf{C}_{\mathsf{awgn}}(P^*) - \frac{1}{2}\big(\delta_1 + \log(\pi e/6)\big)$, then choosing $\varepsilon = \frac{1}{2}\Big(\mathsf{C}_{\mathsf{awgn}}(P^*) - \frac{1}{2}\big(\delta_1 + \log(\pi e/6)\big) - \frac{k}{L}\Big)$ makes the exponent negative, i.e., $P(i)$ will decay exponentially in i. Given $M' \ne M$, let $q(M,M') = \min\{p : m_p \ne m'_p\}$ be the first bit index where they differ. The above argument has $q(M,M') = 1$. In general, while decoding at stage i, at most 2^{jk} distinct M' have $q(M,M') \in \{(i-j)k+1, \ldots, (i-j+1)k-1\}$. In this scenario, the chance that any such M' has $\bar{x}^i(M')$ closer to $\alpha\bar{y}^i$ compared to $\bar{x}^i(M)$ is the same as (10), with i replaced by j. This is because, given that M' and M have the same first $(i-j)k$ bits, their codewords $\bar{x}^{i-j}(M) = \bar{x}^{i-j}(M')$. Thus, the probability of the decoder finding $\bar{x}^i(M')$ closer to $\alpha\bar{y}^i$ is at most $P(j)$. Since Invariant 1 holds for $i = \Omega(\varepsilon^{-2}\log n)$, at any decoding stage i, the chance that any $M' \ne M$ with $i - q(M,M') = \Omega(\varepsilon^{-2}\log n)$ has $\bar{x}^i(M')$ closer to $\alpha\bar{y}^i$ compared to $\bar{x}^i(M)$ is bounded by

$$\sum_{j=1}^{i-\Omega(\varepsilon^{-2}\log n)} P(j) = O\Big(\frac{1}{n^2}\Big), \tag{11}$$

for an appropriately large constant in the $\Omega(\cdot)$ term above. Thus, at any stage of decoding, only messages M' that differ from M in only the most recent $O(\varepsilon^{-2}\log n)$ bits can be closer to $\alpha\bar{y}^i$. But the number of such messages is polynomial in n, with degree depending on ε^{-2}, and $\varepsilon = \frac{1}{2}\Big(\mathsf{C}_{\mathsf{awgn}}(P^*) - \frac{1}{2}\big(\delta_1 + \log(\pi e/6)\big) - \frac{k}{L}\Big)$ (half the *gap*). Therefore, choosing B for the decoder to be this polynomial value, we can ensure that at each stage, the correct message M is one of the candidates. When decoding ends, the remaining candidates are only those that differ from M in the last $O(\varepsilon^{-2}\log n)$ bits.