Verification-Based Decoding for Packet-Based Low-Density Parity-Check Codes

Michael G. Luby and Michael Mitzenmacher, *Member, IEEE*

***Abstract*—We introduce and analyze *verification-based decoding* for low-density parity-check (LDPC) codes, an approach specifically designed to manipulate data in packet-sized units. Verification-based decoding requires only linear time for both encoding and decoding and succeeds with high probability under random errors. We describe how to utilize code scrambling to extend our results to channels with errors controlled by an oblivious adversary.**

***Index Terms*—Low-density parity-check (LDPC) codes, packet- based codes, q-ary symmetric channel, verification-based decoding.**

I. Introduction

WORK on low-density parity-check (LDPC) codes has focused on the scenario where a bitstream is transmitted over a channel that introduces bit-level errors, such as the binary-symmetric error channel with a fixed error probability or with Gaussian white noise (see, e.g., [1]–[6]). For this scenario, encoding and decoding schemes normally perform computational operations on and maintain data structures for individual bits. For example, techniques based on belief propagation [6], [7] use a probability for each bit to represent the current belief that the transmitted bit was a zero or one, and perform computations to update these probabilities. In many practical situations, however, the basic unit of transmission might not be single bits, but blocks of bits organized as packets. Here we use the term packets to broadly refer to a collection of bits. For example, packets could be thousands of bits, as is the case with Internet packets, or a packet could simply represent a 32-bit integer. To achieve high speeds on many current systems, it is natural to consider coding schemes that perform computational operations at the packet level instead of the bit level. For example, packet-based LDPC codes for erasures have been applied to content delivery using reliable multicast and content delivery networks [7]–[10].

A further motivation for studying packet-level schemes is that they can be used in concatenated codes [11]. An inner code that works at the bit level can be used on individual packets. If this inner code fails, the bits in the packet could be erroneous in unpredictable and seemingly arbitrary ways. An outer code designed to deal with errors at the packet level could then be used to correct failures from the inner code. Concatenated codes may allow computationally complex inner codes that work on packets of a small fixed size while also offering scalability to large block lengths using the packet-level outer code. Alternatively, concatenated codes may allow computationally weak inner codes, where a packet-level outer code is used to cope with frequent inner code failure.

In this paper, we introduce and analyze *verification-based decoding*, a decoding approach for simple LDPC codes designed especially for large alphabets. Verification-based decoding is designed to deal with data in packet-sized units of 32 bits or more. More specifically, if the code is over n packets, then the packets should have $\Omega(\log n)$ bits, so that the total alphabet size is suitably large compared to the number of packets. We use the term verification-based decoding because an important aspect of our codes is that packet values are verified as well as corrected through a simple message-passing algorithm.

We first describe verification-based decoding for the case of the q-ary symmetric channel (qSC), for large values of q. In the qSC, the symbols are numbers in the range $[0, q-1]$, and when an error occurs, the resulting symbol is assumed to take on a value uniformly at random from the set of $q-1$ possible incorrect values. We then describe how verification-based decoding for the qSC channel can be applied in more general settings in practice by using code scrambling techniques.

A. Coding for the qSC

In principle, the problem of coding for the qSC should not be very different from that of coding for the q-ary erasure channel (qEC) as q becomes large. As the ratio of their capacities approach 1 as $q \to \infty$, we might expect that there should be some coding scheme that is able to locate errors using relatively little overhead, and then correct the errors using an efficient capacity-approaching erasure-correction code.

Indeed, the standard method of using a c-bit checksum on a b-bit packet to detect packet errors, followed by erasure–correction, can approach capacity as b goes to infinity. The probability that the checksum will not detect an error is 2^{-c}; the overhead is a factor of $c/(b+c)$. Assuming c is chosen large enough that errors will not occur with high probability, low-complexity erasure–correction schemes can be used to approach the capacity of the resulting erasure channel.

Another natural scheme to consider would be a maximum-distance separable (MDS) code over a q-ary alphabet.

Manuscript received September 19, 2003; revised June 8, 2004. The work of M. Mitzenmacher was supported in part by an Alfred. P. Sloan Research Fellowship and the National Science Foundation under Grants CCR-9983832, CCR-0118701, and CCR-0121154. Part of this work was done while M. Mitzenmacher was visiting Digital Fountain, Inc., Fremont, CA. The material in this paper was presented in part at the 40th Annual Allerton Conference on Communication, Control, and Computing, Monticello, IL, October 2002.

M. G. Luby is with Digital Fountain, Inc., Fremont, CA 94538 USA (e-mail: luby@digitalfountain.com).

M. Mitzenmacher is with Harvard University, Division of Engineering and Applied Sciences, Cambridge, MA 02138 USA (e-mail: michaelm@eecs.harvard.edu).

Communicated by R. J. McEliece, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2004.839499

It is easy to show that with maximum-likelihood (ML) decoding, MDS codes can approach the capacity of the qSC as $q \to \infty$. However, ML decoding of known MDS codes such as Reed–Solomon codes is much too complex, and it is not known whether it is possible to construct MDS or MDS-like codes for which low-complexity near-ML decoding schemes exist.

This paper addresses the nonasymptotic but practical cases where q is only moderately large, say from $q = 2^{32}$ to $q = 2^{128}$. In these cases, the c-bit checksum technique may not be effective, because if c is large enough to make the probability of not detecting an error small, the rate overhead will be much too high. We introduce an alternative error-locating scheme based on local checks rather than checksums. Our scheme requires only linear complexity. It achieves respectable rates for moderate-sized q, although it does not approach capacity as $q \to \infty$. We hope that variations on the foundation we present will lead to schemes that do approach capacity.

B. Previous Work

Gallager considered the question of LDPC codes over large alphabets in his seminal work on the subject [13]; however, Gallager did not recognize the full potential of working over the qSC. By making better use of properties of the underlying channel, we achieve significantly better results.

Techniques quite similar to verification-based decoding were suggested by Metzner, using what he calls majority-logic-like decoding [14], [15]. Metzner's results were experimental, and he was the first to suggest that such codes could perform better than Reed–Solomon codes, under certain assumptions. Our work represents a significant advance, in that by placing these codes in the LDPC framework, we are able to give asymptotic analyses of code performance as well as determine improved constructions.

Davey and MacKay [16] develop LDPC codes when the symbols are elements of small finite fields, using belief propagation techniques. Belief propagation is not scalable to the very large alphabets we consider here.

Our work also demonstrates some interesting connections between packet-level error-correcting codes and erasure codes. In this respect, the work has a similar feel to classic work on burst error-correction codes on bit-level channels, where a long bursty error can be treated like a long erasure if it can be detected; see [17] for a survey.

II. Framework for Low-Density Parity-Check Codes

We briefly summarize the now-standard framework for LDPC codes, following [2]. LDPC codes are conventionally represented by bipartite graphs. One on side, the n *variable nodes* correspond to symbols in the codeword. On the other side, the m *check nodes* correspond to constraints on the adjacent variable nodes. We assume henceforth that there are m check nodes. The design rate R is given by $R = \frac{n-m}{n}$. (The actual rate R tends to be slightly higher than the design rate R in practice, because the check nodes are not necessarily all linearly independent. This causes at most a vanishingly small difference as n gets large, so we ignore this distinction henceforth.) The q-ary alphabet is assumed to have an appropriate group structure and group operation. Specifically, the symbols can be interpreted as numbers modulo q, with the constraints being on the sum of values of the variable nodes modulo q. When $q = 2^b$, as when the packet is a string of b bits, we may use the group $(\boldsymbol{Z}_2)^b$, so the sum operation is a bitwise EXCLUSIVE–OR and the constraints are packet-level parity-check constraints.

A family of codes can be determined by assigning degree distributions to the variable and check nodes. In regular LDPC codes, all variable nodes have the same degree, and all check nodes have the same degree. More flexibility can be gained by using *irregular* codes, introduced by Luby, Mitzenmacher, Shokrollahi, and Spielman [2], [7], which allow the degrees of each set of nodes to vary. Let $\vec{\lambda} = (\lambda_2, \ldots, \lambda_{d_v})$ be the vector such that the fraction of edges connected to variable nodes of degree i is λ_i. (We assume a minimum degree of two throughout.) Here d_v is the maximum degree of a variable node. Note that λ_i refers to the fraction of edges, and not the fraction of nodes, of a given degree. Similarly, let $\vec{\rho} = (\rho_2, \ldots, \rho_{d_c})$ be such that the fraction of edges connected to check nodes of degree i is ρ_i, and d_c is the maximum degree of a check node. Based on these degree sequences, we define the polynomials

$$\lambda(x) := \sum_{i=2}^{d_v} \lambda_i x^{i-1} \quad \text{and} \quad \rho(x) := \sum_{i=2}^{d_c} \rho_i x^{i-1}$$

which prove useful in subsequent analysis. The λ_i and ρ_i variables must satisfy a constraint so that the number of edges is the same on both sides of the bipartite graph. This constraint is easily specified in terms of the design rate by the equation

$$R = 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx}.$$

Once degrees have been chosen for each node (so that the total degree of the check nodes and the variable nodes are equal), a specific random code can be chosen by mapping the edge connections of the variable nodes to the edge connections of the check nodes. That is, to select a code at random, a random permutation π of $\{1, \ldots, E\}$ is chosen, where E is the number of edges. For all $i \in \{1, \ldots, E\}$, the edge with index i out of the left side is identified with the edge with index π_i out of the right side. In practice, it appears better not to choose the edge connections entirely randomly. For example, improvements generally occur by avoiding duplicate edges between pairs of nodes or short cycles in the graph. We do not concern ourselves with these issues in the subsequent analysis.

Unless otherwise specified, we assume that the constraints are such that the sum of the symbols associated with the variable nodes adjacent to each check node is 0. In some circumstances, it may be better to design a layered code, as described in [2]; this does not affect the analysis. For the layered version, the encoding time is proportional to the number of edges in the graph. Linear time encoding schemes also exist for the single-layer scheme we analyze here [18].

We consider message-passing algorithms described below. To determine the asymptotic performance of such codes, it suffices to consider the case where the neighborhood of each node is a

tree for some number of levels. That is, there are no cycles in the neighborhood around each node. Analysis in this case is greatly simplified since random variables that correspond to messages in our message-passing algorithms can be treated as independent. The analysis is accurate because the graph is asymptotically tree-like, and can be made rigorous using now-standard martingale arguments, as introduced in [19].

III. Decoding Algorithms for the qSC

A. A Simple Decoding Algorithm

We suggest several decoding algorithms for the qSC, first presenting a very simple algorithm that we improve subsequently. Before explaining how to view our decoding algorithm as a message-passing algorithm, we first describe an equivalent algorithm from the point of view of the nodes for clarity.

With each variable node v there corresponds a true value t_v, a received value r_v, and a current value c_v. Throughout the algorithm, each variable node is in one of two possible states: either *unverified* or *verified*. When a node is unverified, the algorithm has not yet fixed the final value for that node. Hence, the decoding algorithm begins with all nodes being unverified. When a node is verified, its current c_v becomes fixed. Hence, the algorithm should, with high probability, never assign a verified node a value c_v such that $c_v \neq t_v$. In the algorithm that follows, the current value c_v is always equal to r_v when the node is unverified.

The decoding algorithm simply applies the following rules in any order as much as possible.

1) If the sum of the current values of all the neighbors of a check node equals 0, all currently unverified neighbors become verified and the final values are fixed to the current values.
2) If all but one of the neighbors of a check node is verified, the remaining neighbor becomes verified, with its final value being set so that the sum of all neighbors of the check node equals 0.

In order to analyze this algorithm, we refine our description of the node state. An unverified node is *correct* if its value was received correctly, and *incorrect* if it was received in error. This refinement is used only in the analysis, and not in the decoding algorithm, which does not know if an unverified node is correct or incorrect. Also, recall that we assume that an incorrect packet takes on a uniform incorrect value.

In this decoding process, the check nodes play two roles. First, they may *verify* that all of their neighbors are correct, according to Rule 1. This verification rule applies because of the following fact: if the sum of the current values of all neighbors of a check node is 0, then with high probability all the neighboring variable nodes must be correct. Concretely, we have the following lemma.

Lemma 1: At any step where a check node attempts to verify all of its neighbors, the probability of an error is at most $1/(q-1)$. Over the entire decoding algorithm, if C verification steps are attempted, the probability of an error is at most $C/(q-1)$.

Proof: For an erroneous verification to occur, two or more neighbors of a check node c must be in error. Consider an arbitrary neighbor in error, v. Given the values of the other neighbors of c, there is at most one possible erroneous value for v that would lead to a false verification. Under the assumption that errors take on an incorrect value that is uniform over all $q-1$ possibilities, the probability that v takes on this precise value is at most $1/(q-1)$. Hence, at each step where a check node attempts to verify all of its neighbors, the probability of an error is at most $1/(q-1)$. The second statement of the lemma then follows from the first by a simple union bound. □

To see the value of the preceding lemma, consider the case where symbols consist of b bits, so $q = 2^b$. The probability of a failure from a false verification is exponentially small in b at each step. It is straightforward to design a decoding process so that the number of verification steps attempted is linear in the size of the graph. Initially, all check nodes may gather the received values of their neighboring variable nodes, and see if a verification is possible. A check node can also take action whenever the state of one of its neighboring nodes changes. The total work done is then proportional to the number of edges in the graph. For bounded degree graphs, the number of edges will be $O(n)$. Hence, the packet size b needs to be only $\Omega(\log n)$ bits in order that the probability of failure due to a false verification be polynomially small in n. Specifically, packets of size $b = 32$ bits should suffice for codes with hundreds or thousands of nodes, and packets of size $b = 64$ bits should suffice for codes with tens or hundreds of thousands of nodes.

The other role of a check node is to correct a neighboring variable node that was received incorrectly, according to Rule 2. A check node can correct a neighbor after all other neighbors have been verified and therefore are known (with high probability) to have the correct value. In this case, the value of the unverified neighbor is obtained by determining the value that results in a 0 sum at the check node.

B. A Message-Passing Decoding Algorithm

We now develop a message-passing version of this decoding process to aid our analysis. The goal is to determine the asymptotic error threshold p^*, which is the limiting fraction of errors tolerable under our decoding process as n grows large. To picture the decoding process, we focus on an individual edge (v, c) between a variable node v and a check node c, and an associated tree describing the neighborhood of v. Recall that we assume that the neighborhood of v is accurately described by a tree for some fixed number of rounds. The tree is rooted at v, and the tree branches out from the check nodes of v excluding c, as shown in Fig. 1.

Given that our graph is chosen at random, we can specify how this tree branches in a natural way. This specification is the approximation obtained by thinking of the tree growing from v as a branching process, which is correct in the asymptotic limit as the number of nodes grows to infinity. As the probability that edge (v, c) has degree j is λ_j, with probability λ_j there are $j-1$ other check node neighbors of v. Similarly, every such neighbor c' has $j-1$ other variable node neighbors with probability ρ_j, and so on down the tree.

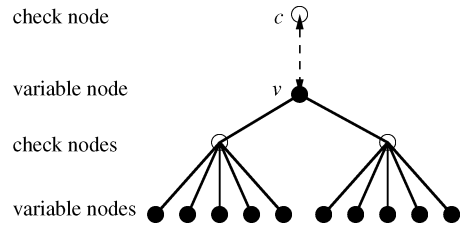

Fig. 1. The neighborhood around (v, c).

We think of the decoding process as happening in rounds, with each round having two stages. In the first stage, each variable node passes to each neighboring check node in parallel its current value and state. In the second stage, each check node c' sends to v a flag denoting whether it should change its state to verified; if c' verifies v, it also sends the appropriate value. Based on this information, v changes its value and state appropriately. For convenience in the analysis, we think of each variable node as passing on to the check node c the current value excluding any information obtained directly from c. (This avoids the problem of a circular flow of information.) That is, when the variable node v passes information to c regarding its value and state, it only considers changes in its state caused by other nodes.

We provide an analysis based on the tree model. Consider an edge (v, c) in the graph. Let a_j be the probability that in round j the message from v to c contains the true value t_v but v is unverified. Similarly, let b_j be the probability that in round j the message from v to c contains an incorrect value for v. We ignore the possibility in the analysis that a false verification occurs, since as we have already argued, for a sufficiently large alphabet this occurs with negligible probability. Hence, $1-a_j-b_j$ is the probability that, in round j, v can confirm to c that it has been verified via another check node. Initially a_0 is simply the initial probability a correct word is sent and $b_0 = 1 - a_0$. If $a_j + b_j$ tends to 0, then our decoding algorithm will be successful, since then the probability that an edge (and therefore its corresponding node) remains unverified falls to 0.

The evolution of the process from round to round, assuming that the neighborhood of v is given by a tree, is given by

$$a_{j+1} = a_0\lambda(1 - \rho(1 - b_j)) \quad (1)$$
$$b_{j+1} = b_0\lambda(1 - \rho(1 - a_j - b_j)). \quad (2)$$

We explain the derivation of (2) by considering the decoding from the point of view of the edge (v, c). For an incorrect value to be passed in the $(j+1)$th round, the node v must have been received incorrectly; this corresponds to the factor b_0. Also, it cannot be the case that there is some check node c' other than c neighboring v that has all of its children verified after j rounds, or else v could be corrected and verified for the $(j+1)$th round. Now each c' has $k-1$ children below it with probability ρ_k, and each child is verified after j rounds with probability $1-a_j-b_j$. The probability that v has not been corrected due to a specific check node c' by round j is therefore

$$\sum_i \rho_i(1 - a_j - b_j)^{i-1} = \rho(1 - a_j - b_j).$$

As v has $k-1$ other neighboring check nodes besides c with probability λ_k, the probability that v remains uncorrected when passing to node c in round $j+1$ is

$$\sum_i \lambda_i(1 - \rho(1 - a_j - b_j))^{i-1} = \lambda(1 - \rho(1 - a_j - b_j)).$$

This yields (2); (1) is derived by similar considerations.

We show how to use this analysis to find codes with good properties. We first modify the above equations as follows:

$$a_{j+1} = a_0\lambda(1 - \rho(1 - b_j)) \quad (3)$$
$$b_{j+1} = b_0\lambda(1 - \rho(1 - a_{j+1} - b_j)). \quad (4)$$

Here (4) differs from (2) in that we have replaced a_j with a_{j+1}. This change does not change the final performance of the decoding; intuitively, this change is equivalent to changing our processing at each round by splitting it into two subrounds. In the first subround, variable nodes that have the correct value have their state updated. In the second subround, variable nodes with an incorrect value are corrected. The split clearly does not affect the final outcome; however, it allows us to replace a_j with a_{j+1} in the defining equation for b_j.

With this change, we find

$$b_{j+1} = b_0\lambda(1 - \rho(1 - (1 - b_0)\lambda(1 - \rho(1 - b_j)) - b_j)). \quad (5)$$

We have reduced the analysis to an equation in a single family of variables b_j. It is clear that if b_j converges to 0, then so does a_j, by the definition of the decoding process. Hence, we need only consider the b_j. For $b_j \to 0$ with initial error probability b_0, we require that the sequence of b_j decrease to 0. In particular, it suffices to find $\lambda(x)$ and $\rho(x)$ so that

$$b_0\lambda(1 - \rho(1 - (1 - b_0)\lambda(1 - \rho(1 - x)) - x)) < x \quad (6)$$

for $0 < x \leq b_0$. If we let $x = b_j$ in (6) above, we find that it corresponds to $b_j < b_{j+1}$, and, in fact, it implies that the sequence of b_j decrease to 0. Based on our discussion, we have the following theorem.

Theorem 1: Given a design rate R and an error probability b_0, then if there are $\vec{\lambda}$ and $\vec{\rho}$ satisfying the rate constraint

$$R = 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx}$$

and the code constraint

$$b_0\lambda(1 - \rho(1 - (1 - b_0)\lambda(1 - \rho(1 - x)) - x)) < x$$

for $0 < x \leq b_0$, then for any $\epsilon > 0$ there are codes of rate R that can correct errors on a qSC with error probability $b_0 - \epsilon$ with high probability using the verification-based decoding scheme described earlier.

We emphasize that similar theorems using various decoding schemes are implicit throughout the rest of the paper. Equation (5) therefore provides us a tool for determining good sequences $\vec{\lambda}$ and $\vec{\rho}$. This is a nonlinear equation in the coefficients λ_j and

ρ_j, yielding a nonlinear optimization problem for which standard numerical techniques can be applied.

Unfortunately, solving the nonlinear optimization problem directly for a specific code rate does not shed a great deal of insight into what can be said for general code rates. We can, however, demonstrate a provable bound for this family of codes based on the family of codes determined in [2] for erasures. (Other constructions of codes with similar properties have been subsequently developed, for example, in [20], [21].)

Lemma 2: For any $0 < R < 1$ and $\epsilon > 0$, there exist $\vec{\lambda}$ and $\vec{\rho}$ corresponding to a family of erasure codes of rate R that correct a $(1-R)(1-\epsilon)$ fraction of errors with high probability such that

$$\lambda(1-\rho(1-x)) < \frac{x}{(1-R)(1+\epsilon)}, \qquad 0 < x \leq 1. \quad (7)$$

This lemma allows the following theorem.

Theorem 2: For any $0 < R < 1$ and $\epsilon > 0$, there exists a family of codes of rate R that correct a

$$1 - \frac{R}{2} - \frac{\sqrt{4R-3R^2}}{2} - \epsilon$$

fraction of errors with high probability using verification-based decoding.

Proof: We use the $\vec{\lambda}$ and $\vec{\rho}$ defined by the erasure codes of Lemma 2, and apply the corresponding inequality to find the asymptotic fraction of errors we can correct using verification-based decoding for these degree sequences. Let $\gamma = (1-R)(1+\epsilon)$. We seek the maximum value of b_0 for which

$$b_0\lambda(1-\rho(1-(1-b_0)\lambda(1-\rho(1-x))-x)) < x.$$

By repeatedly using Lemma 2, we find

$$\begin{aligned} b_0\lambda(1-\rho(1-(1-b_0)\lambda(1-\rho(1-x))-x)) &< \frac{b_0((1-b_0)\lambda(1-\rho(1-x))+x)}{\gamma} \\ &< \frac{\frac{b_0(1-b_0)x}{\gamma} + b_0 x}{\gamma} \\ &= x\frac{b_0(1-b_0)+b_0\gamma}{\gamma^2}. \end{aligned}$$

Here, the first inequality follows by applying Lemma 2 to the outer $\lambda(1-\rho(1-z))$ expression, and the second inequality similarly follows by then applying it to the inner expression. The final right-hand size is less than or equal to x whenever we choose

$$\frac{b_0(1-b_0)+b_0\gamma}{\gamma^2} \leq 1$$

and solving this quadratic we find we may choose

$$b_0 \leq \frac{1+\gamma-\sqrt{1+2\gamma-3\gamma^2}}{2}. \quad (8)$$

As γ can be arbitrarily close to $1-R$, asymptotically there exist codes that can correct anything less than a fraction $1 - \frac{R}{2} - \frac{\sqrt{4R-3R^2}}{2}$ of errors with high probability using verification-based decoding. □

As a comparison, we might consider Reed–Solomon codes, which in the worst case can correct up to a fraction $(1-R)/2$ errors very efficiently in practice. We find

$$1 - \frac{R}{2} - \frac{\sqrt{4R-3R^2}}{2} > (1-R)/2$$

when $R < 1/3$. Hence, when $R < 1/3$, the verification-based decoding of Theorem 2 outperforms the worst case bounds for Reed–Solomon codes. This is admittedly a somewhat unfair comparison, since verification-based decoding corrects random errors, and the result above applies to worst case errors for Reed–Solomon codes. As discussed in Section I-A, in theory, for the qSC, Reed–Solomon codes can approach the capacity $1-R$ as q goes to infinity, but we know of no such scheme that requires only linear decoding time or for q to be only polynomial (as opposed to exponential) in n. While verification-based decoding does not meet the capacity $1-R$, its computational complexity and the utility for packets with small numbers of bits b is appealing.

IV. Improvements

A. Additional Verification

We may improve our verification-based decoding by allowing further means of verification. We describe the changes to the message-passing algorithm. In the first stage, each variable node passes to each neighboring check node its current value and state. In the second stage, each neighboring check node c' sends to v a flag denoting whether c' can verify v directly, using one of the two rules given previously. If so, c' again sends the verified value as before. If not, c' also sends to v a *proposed value*, which is the value that v should take if all of the other neighbors of c' have sent the correct value. Now suppose v receives two proposed values that are the same. In this case, v should change its value to the proposed value and label itself as verified for the next round.

The reasoning behind this improvement is similar to the original argument for verification. If all neighbors besides v for a check node c' are in fact correct, the proposed value will be the correct value for v. If not, the proposed value will be random over all incorrect possibilities, and hence the probability of a match is small. We must adopt the additional restriction that there are no cycles of length four in the bipartite graph that represents the code, however. Otherwise, two check node neighbors of v could be neighbors with the same variable node v_2; if v_2 is in error, the proposed values of the check nodes would match, inducing a subsequent incorrect verification at v.

This improvement does increase the probability of a false verification, since there are now many additional ways a false verification could occur. The number of verification steps is proportional to the sum of the squares of the degrees of the variable nodes, instead of to the sum of the degrees of the variable nodes, as in the original scheme. Assuming a constant maximum degree, $b = \Omega(\log n)$ bits per packet still ensures that no false verification occurs with high probability.

The resulting equations describing the asymptotic behavior in this situation are somewhat more difficult. Again we have

$$a_{j+1} = a_0\lambda(1-\rho(1-b_j))$$

by the same reasoning as before. To determine an equation for b_{j+1}, note that for v to continue to hold an incorrect value to pass to c, one of the following events must occur:

- all neighbors of v other than c received an incorrect value from some other neighbor in the previous round; or
- all but one neighbor of v other than c received an incorrect value from some other neighbor in the previous round, and the one neighbor that received all correct values did not have all of these values verified.

The probability that the first case occurs is just $\lambda(1-\rho(1-b_j))$. For the second case, when v has $i-1$ other neighbors, the probability that a specific set of $i-2$ neighbors other than c receive at least one incorrect value during round j is $(1-\rho(1-b_j))^{i-2}$. Ignoring (temporarily) the probability that the last neighbor sends a correct value, since there are $i-1$ possible ways of choosing the correct neighbor, we have the term

$$\sum_i \lambda_i (i-1)(1-\rho(1-b_j))^{i-2} = \lambda'(1-\rho(1-b_j))$$

where λ' is the derivative of λ. We multiply this term by the probability that the last check node sends the correct but unverified proposed value, which is $(\rho(1-b_j)-\rho(1-a_j-b_j))$. (Here $\rho(1-b_j)$ is the probability of that all of the other neighbors of the last check are correct, and $\rho(1-a_j-b_j)$ is the probability that all of the other neighbors are verified.) Putting this all together yields

$$b_{j+1} = b_0\left[\lambda(1-\rho(1-b_j)) + \lambda'(1-\rho(1-b_j))(\rho(1-b_j)-\rho(1-a_j-b_j))\right]. \quad (9)$$

Again, we can modify the equation for b_j into one that does not involve a_j by replacing a_j by a_{j+1} and substituting the equation for a_{j+1} in (9).

$$b_{j+1} = b_0\left[\lambda(1-\rho(1-b_j)) + \lambda'(1-\rho(1-b_j))(\rho(1-b_j) - \rho(1-(1-b_0)\lambda(1-\rho(1-b_j))-b_j))\right]. \quad (10)$$

Finding codes of a specified rate that asymptotically handle a fraction b_0 of errors now corresponds to finding an appropriate $\lambda(x)$ and $\rho(x)$ satisfying the rate condition and

$$x > b_0\left[\lambda(1-\rho(1-x)) + \lambda'(1-\rho(1-x))(\rho(1-x) - \rho(1-(1-b_0)\lambda(1-\rho(1-x))-x))\right]. \quad (11)$$

Equation (11) corresponds to a design criterion for codes using verification-based decoding under the described decoding algorithm. Again, attempting to maximize the tolerable error probability for a given rate is a complex nonlinear optimization problem that we do not delve into here. Also, we have not developed a theorem similar to Theorem 2, because of the challenging form of (10). Because the decoding rules are a superset of those of the original scheme, a greater fraction of errors can be tolerated. The improvement can be nontrivial; for example, for a regular rate $1/2$ code with all nodes having degree 3 on the left and degree 6 on the right, the fraction of tolerable errors jumps from approximately 17% to approximately 21%. The price, as we have mentioned, is a higher probability of a false verification, which is still negligible for a sufficiently large number of bits per packet.

The decoding algorithm we have described is still, in some sense, rather conservative. A variable node repeatedly passes on its received value until it is corrected. At each step, each variable node receives a proposed value from each neighboring check node, but nothing is done with these proposed values unless some two of them match. One might extend such a scheme so that each variable node keeps track of a list of proposed values received each round and passes this entire list on to the check nodes. The check nodes can then determine if any possible set of values within the cross-product of proposed values of its neighbors appropriately satisfy the check node, and respond accordingly. The advantage of such a scheme is that allowing proposed values to propagate and be verified may lead to more verifications, more quickly. The problem with such a scheme is that the lists of proposed values may become too long. This may cause a great deal of additional computation, since the cross-product of all proposed values from the message nodes would be considered at a check node. Additionally, this approach greatly increases the probability that an erroneous verification occurs, since the number of possible combinations grows with the product of the lengths of the lists. We leave the possibility of designing looser verification codes that pass more extensive information for exploration in future work.

B. Using Reed–Solomon Codes With Verification-Based Decoding

We may vary our codes by allowing the check nodes to hold information other than the sum of the variable nodes. A natural approach is to constrain k-tuples of variables to belong to an Reed–Solomon code with more than one redundant symbol, and store these redundant symbols in the check nodes. For example, consider the case where pairs of check nodes are associated with k variable nodes, so that the check node values are calculating using a $(k+2,k,1)$ Reed–Solomon code. The check nodes then have the property that they can correct any single error among the k neighboring variable nodes. As we explain later, the check nodes have further correction properties; however, it is instructive to consider a decoding scheme which *just* uses the above property.

Consider our message-passing decoding scheme in this setting. For any pair of check nodes and the associated set of variable nodes, if there is a single error among the variable nodes, it can be corrected and all the associated variable nodes verified. Again, we have used here our assumption that an error replaces a value with a value taken uniformly at random. Because of this assumption, with high probability throughout the execution of the decoding no set of variable nodes containing multiple errors will be falsely verified with high probability.

In the message-passing algorithm, a variable node v will transmit an incorrect value to a neighboring check node pair c after $j+1$ rounds if and only if it was received incorrectly and every other neighboring check node pair had another incorrect message node neighbor after round j. This yields the following recurrence equation to describe b_j:

$$b_{j+1} = b_0\lambda(1-\rho(1-b_j)). \quad (12)$$

This is exactly the same recurrence for the erasure codes developed in [7]. By using Reed–Solomon codes, we are making

errors "equivalent" to erasures in the LDPC code setting. As described in [7, Lemma 2], vectors $\vec{\lambda}$ and $\vec{\rho}$ were determined that are essentially optimal: for codes of rate R we can come arbitrarily close to the optimal tolerable loss probability $1 - R$. We can apply these distributions (or other essentially optimal distributions, as described in [20], [21]) to obtain the following theorem.

Theorem 3: For any $0 < R < 1$ and $\epsilon > 0$, there exist sequences $\vec{\lambda}$ and $\vec{\rho}$ corresponding to a family of codes of rate R such that verification-based decoding with Reed–Solomon codes as described above corrects a $(1 - R)/2 - \epsilon$ fraction of errors with high probability.

Proof: We use the $\vec{\lambda}$ and $\vec{\rho}$ for an erasure code of rate $(1 + R)/2$ as guaranteed by Lemma 2. Such a code corresponds to a rate R code in this setting since each check node uses two Reed–Solomon values per node. (To see this, consider that a rate $(1 + R)/2$ erasure code with n message nodes has $n(1 - R)/2$ check nodes, which in this case corresponds to $n(1 - R)$ symbol values.) Since we obtain the same recurrence (12) for these codes as for the erasure codes, we can correct a $(1 - R)/2 - \epsilon$ fraction of errors with high probability. □

We find that under this simple decoding rule we can correct almost the same fraction of errors as Reed–Solomon codes under the worst case. Again, we emphasize the caveat that our results hold only asymptotically with high probability for the qSC.

The decoding scheme above can be improved slightly. We are not taking advantage of the following additional power of the check nodes: whenever a check node pair has at most two unverified neighbors, the remaining neighbors can be corrected, as the correct and verified values suffice to reconstruct the remaining ones. A decoding scheme that takes advantage of this fact is not substantially more complex, although the equations that describe it are. Consider again a recursive description of the b_j. It is now not enough that every neighboring check node pair of a variable node v has another incorrect variable node neighbor; now it must also have at least one other unverified neighbor.

Consider the probability that after round j a check node pair c' neighboring v has one other incorrect variable node neighbor but no other unverified neighbors. For a pair with k neighbors, we must choose the one other incorrect neighboring message node, and all other nodes are verified. Hence, this probability is

$$\sum_{k \geq 2} \rho_k (k-1) b_j (1 - a_j - b_j)^{k-2}$$

which equals $b_j \rho'(1 - a_j - b_j)$, where ρ' is the derivative of the ρ. This yields the recurrence

$$b_{j+1} = b_0 \lambda(1 - \rho(1 - b_j) - b_j \rho'(1 - a_j - b_j)). \tag{13}$$

A similar formula for the a_j values can be obtained. Here, a variable node with the correct value will become verified if there is a neighboring check node pair c' such that all other variable node neighbors of c' have the correct value, or if there is a neighboring check node pair c' with only one erroneous neighbor. This yields the recurrence

$$a_{j+1} = a_0 \lambda(1 - \rho(1 - b_j) - b_j \rho'(1 - b_j)). \tag{14}$$

Following our standard practice, we can combine these into a single equation that can be used as a design criterion for good sequences.

$$b_{j+1} = b_0 \lambda(1 - \rho(1 - b_j) - b_j \rho'(1 - (1 - b_0)\lambda(1 - \rho(1 - b_j) - b_j \rho'(1 - b_j)) - b_j)). \tag{15}$$

Similar equations can be developed for Reed–Solomon codes that can correct more errors. If the check nodes are associated with k variable nodes, and the check node values are calculated using a $(k + \ell, k, \ell/2)$ Reed–Solomon code (where ℓ is an even integer), then

- when a check node collection c' has at most $\ell/2$ neighboring variable nodes that are in error, it can correct and verify all of its neighbors, and
- when a check node collection c' has at most ℓ neighboring variable nodes that are unverified, and all other neighboring nodes are verified, it can correct and verify all of its erroneous neighbors.

The appropriate recurrences are

$$a_{j+1} = a_0 \lambda \left(1 - \sum_{i=0}^{\ell/2} \frac{b_j^i \rho^{(i)}(1 - b_j)}{i!} \right) \tag{16}$$

$$b_{j+1} = b_0 \lambda \left(1 - \sum_{i=0}^{\ell/2-1} \frac{b_j^i \rho^{(i)}(1 - b_j)}{i!} - \sum_{i=\ell/2}^{\ell-1} \sum_{k=0}^{i-\ell/2} \frac{a_j^k b_j^{i-k} \rho^{(i)}(1 - a_j - b_j)}{k!(i-k)!} \right). \tag{17}$$

Here $\rho^{(i)}(x)$ is the ith derivative of ρ at x, with the 0th derivative just being ρ itself.

V. Code Scrambling

Up to this point, we have assumed that our codes function on a qSC. This is a rather strong assumption; one might expect that errors introduced in packets are correlated in some nontrivial way. For example, perhaps it is always the last bit that is changed. In this case, verification-based decoding as we have described will perform poorly, since the sum of two erroneous packets will be the same as two correct packets, and hence the analysis of Lemma 1 will not apply. In general, we cannot know ahead of time what types of correlated errors might be introduced by the channel.

It is well known to data communications engineers that any q-input, q-output channel may be reduced to a memoryless qSC by randomization techniques such as scrambling and interleaving. In this section, we discuss this idea in the framework of [12], which gives the precise theorem we will use. (We rediscovered this theorem ourselves independently.)

Suppose the sender and receiver have a source of shared random bits. We model the errors introduced by the channel as being governed by an oblivious adversary, who is unaware of the random bits shared by the sender and receiver. In this setting, the sender and receiver can use the random bits to ensure that regardless of the strategy of the adversary, the errors introduced appear equivalent to those introduced by a qSC, in the following sense: if the adversary modifies d transmitted

symbols, the effect is as though d randomly selected transmitted symbols take on erroneous values taking on random values from GF(q).

The sender and receiver use their shared random bits as follows. When sending values $x_1, \ldots, x_n$, the random bits are used to determine values $a_1, \ldots, a_n$ chosen independently and uniformly at random from the nonzero elements GF$(q)^*$ and values $b_1, \ldots, b_n$ chosen independently and uniformly at random from GF(q). Instead of sending the symbol x_i, the sender sends the symbol $a_i x_i + b_i$. Further, before sending the modified symbols, a permutation π on $\{1, \ldots, n\}$ is chosen uniformly at random and the symbols are sent in the order given by the permutation.

Informally, it is clear that these steps stifle the adversary. Since each symbol is now equally likely to take on any value from GF(q), after the permutation of the symbols to be sent the adversary cannot distinguish the original position of any transmitted symbol. Hence, if the adversary modifies d symbols, their locations will appear random to the receiver after the permutation is undone. Further, suppose the adversary adds an error e_i to the transmitted symbol $a_i x_i + b_i$, so that the received value is $a_i x_i + b_i + e_i$. The receiver will reverse the symbol transformation, obtaining the symbol $x_i + (a_i)^{-1} e_i$ in place of x_i. Since a_i is uniform over GF$(q)^*$ and unknown to the adversary, the erroneous symbol appears to take on a random erroneous value over GF(q). A more formal proof appears in [12]. Specified to our situation, we have the following.

Theorem 4: The probability p that an adversary who introduces d errors when using code scrambling as described above causes a decoding error is equal to the probability that the verification code makes an error when d errors are introduced in the qSC.

In practice, a truly large sequence of random bits would not be necessary. A pseudorandom generator of small complexity would perform adequately, as errors introduced in real channels are not generally adversarial and specifically would almost surely be independent of the pseudorandom generator.

VI. Conclusion

Verification-based decoding with code scrambling demonstrates that the power of LDPC codes can be exploited for handling errors at the packet level. Using the simple paradigm that a check node can verify when its neighbors are correct in the case where errors are random and the alphabets are sufficiently large, verification-based decoding allows for extremely fast and simple packet-level encoding and decoding algorithms. By making use of code scrambling, verification-based decoding can be made to apply in settings even where errors may not be random.

The most important open problem is to gain more insight into the equations that arise in the analysis of verification-based decoding in order to design optimal families of codes from them. We have shown that by using families of codes that are known to be good for erasures we can prove that verification-based decoding can perform as well as the worst case bounds for Reed–Solomon codes, which in itself is an interesting result. These families of codes were not designed specifically for the equations that arise from verification-based decoding, however, so better bounds are clearly possible. The goal remains to use verification-based decoding to approach the capacity as q grows large.

The framework for verification-based decoding may also apply to other settings. For example, the second author has described codes for packet-based deletion channels using this framework that can be encoded and decoded in polynomial time [22].

References

[1] S. Chung, G. D. Forney, T. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[2] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 585–598, Feb. 2001.

[3] D. J. C. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[4] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 32, pp. 1645–1646, 1996.

[5] T. Richardson, M. A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[6] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[7] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 569–584, Feb. 2001.

[8] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," in *Proc. ACM SIGCOMM '02*, vol. 32, 2002, pp. 47–60.

[9] J. W. Byers, M. Luby, and M. Mitzenmacher, "Accessing multiple mirror sites in parallel: Using Tornado codes to speed up downloads," in *Proc. IEEE INFOCOM '99*, Mar. 1999, pp. 275–283.

[10] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 8, pp. 1528–1540, Oct. 2002.

[11] G. D. Forney, Jr., *Concatenated Codes*. Cambridge, MA: MIT Press, 1966.

[12] P. Gopalan, R. Lipton, and Y. Z. Ding, "Codes, adversaries, and information: A computational approach," paper, 2001, submitted for publication.

[13] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[14] J. M. Chung and J. J. Metzner, "Theoretical analysis of the error correction performance of majority-logic-like vector symbol codes," *IEEE Trans. Commun.*, vol. 49, no. 6, pp. 979–987, Jun. 2001.

[15] J. J. Metzner, "Majority-logic-like decoding of vector symbols," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1227–1230, Oct. 1996.

[16] M. C. Davey and D. J. C. MacKay, "Low density parity check codes over GF(q)," *IEEE Commun. Let.*, vol. 2, no. 6, pp. 165–167, Jun. 1998.

[17] G. D. Forney, Jr., "Burst-correcting codes for the classic bursty channel," *IEEE Trans. Commun. Technol.*, vol. COM-19, no. 5, pp. 772–781, Oct. 1971.

[18] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[19] M. Luby, M. Mitzenmacher, and M. A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *Proc. 9th Annu. ACM-SIAM Symp. Discrete Algorithms*, 1998, pp. 364–373.

[20] M. A. Shokrollahi, "New sequences of linear time erasure codes approaching the channel capacity," in *Proc. 13th Int. Symp. Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes (Lecture Notes in Computer Science)*. Berlin, Germany: Springer-Verlag, 1999, vol. 1719, pp. 65–76.

[21] P. Oswald and M. A. Shokrollahi, "Capacity-achieving sequences for the erasure channel," *IEEE Trans. Inf. Theory*, vol. 48, no. 12, pp. 3017–3028, Dec. 2002.

[22] M. Mitzenmacher, "Verification codes for deletions," in *Proc. 2003 IEEE Int. Symp. Information Theory*, Yokohama, Japan, p. 217.