

# Simplification of packet-symbol decoding with deletions, mis-ordering of packets, and no sequence numbers

John J. Metzner

**Abstract.** A new method is described which builds on Mitzenmacher's idea of adding a different pseudo-random number to each packet to help in verification-based decoding low density codes with packet deletions or errors, or out-of-order receptions, and no sequence numbers. The new method has less decoding complexity and is applicable to any parity check code structure, not just low density codes. Performance depends on the weight distribution of the code. For deletions with ordered reception, the method tolerates, roughly, up to only one more deletion, or packet error, than an erasure channel that knows the positions of all packet errors or deletions. With out-of-order reception, it tolerates, roughly, up to only two more deletions than the erasure channel. Also, it is shown how convolutional codes can further simplify decoding through a partition of the verification search. Moreover, with a modest amount of error detection built into the data, most of the deficiency relative to the erasure channel can be overcome by using error detection to choose from a small list of possibilities. **Index terms:** packet-symbol decoding, deletion codes, verification-based decoding

## I. INTRODUCTION

With the growing amounts of data transfer, there has been recent interest in use of redundant packets as part of a packet-based decoding scheme. Work appeared first in relation to multicasting [1-4], where extra packets called "repair" packets are sent, either proactive, where additional redundant packets are sent initially, or reactive, where redundant packets are sent as needed. The redundant packets can be computed according to a Reed-Solomon code, but a simpler method of vector XOR of packets may be preferred. Multicast work emphasized erasure decoding, where, by packet error detection and sequence numbering, packets are either known as correct or erased.

The idea of tornado codes that are low density parity check codes for the erasure channel was described in [4,5]. A related, but limited idea about using short constraint length convolutional codes for the erasure channel appears in [6]. Tornado codes are simple to decode and effective.

In [7], the use of packet-based low density codes for correcting errors on the q-ary symmetric channel using the verification concept was described, and achievability of rates close to capacity was reported in [8]. The value of q is usually  $2^b$ , where b may be the number of bits in a packet. The verification idea is related to the "declared correct" idea described in [9] for majority-logic-like decoding. Also, the concept of vector symbol decoding [10,11] deals with multi-bit symbols and

relies on finding vector sets adding to zero, suggesting the vectors in the sets are likely to be correct.

In [12,13], Mitzenmacher presented an idea for decoding low density packet-based codes with deletions, without the use of sequence numbers. The approach in [13] is to add a different pseudo-random vector to each packet. This idea is related to the idea in [14] of adding a “watermark string” to a sequence of inner symbols to help mark position in case of deletion or rearrangement. Another related work [15] makes use of Justesen’s approach [16] of using a different inner code for each symbol of a concatenated code. This also provides a means of identifying symbol location in the overall code word. However, the methods in this paper and in [13] use no code in the packet, as well as no sequence numbers. There can be some benefit to including a few error detection bits in a packet, however, as will be discussed.

Systems using packet sizes of less than about 200 bits can benefit substantially in overhead efficiency by using no sequence numbers and limited packet error detection. Decoding complexity in [13] is polynomial in code length  $n$  packets, but complexity is substantially greater than with an erasure channel. Because of the need to try many combinations to find the positions to verify, complexity is a rather high power of  $n$ . For an example given with a modest  $n$  value and code rate  $1/2$ , a complexity of  $O(n^6)$  is stated. For general rate  $R = (1-\epsilon)C$ , where  $C = 1-p$  for the binary erasure channel with erasure probability  $p$ , the complexity is said to be  $O(n^z)$ , where  $z$  is  $O(1/\epsilon^2)$ . Rates closer to capacity require a high exponent of polynomial complexity.

The new method to be described in this paper builds on Mitzenmacher’s idea of using no sequence numbers and, but uses a different approach that allows discovery and proper placement of all correct packets more simply, without trying combinations. The packets can be received out of order and may be mixed with packets that are foreign to the code, multiple copies of a packet, or packets with errors. Complexity for the basic form of the method is approximately  $O(n^2)$  if an  $n$ -component column vector XOR counts as one operation. The basic method requires packet length  $> n$ , but removing sequence numbers might not be justified for large  $n$ . Modifications using convolutional codes or related structures can partition the search for verification sets. This can remove the need for packet length  $> n$  and reduce complexity.

An interesting discovery is that any binary code with a minimum distance greater than 2 can be used. Better minimum distance is preferred for usually better performance. Effectiveness is only slightly inferior, for the same code, to where all incorrect positions are known (erasure channel). For the ordered case, it tolerates about one less deletion than the erasure channel, and with out-of-order

reception it tolerates about two less deletions than the erasure channel. Also, it has the advantage of turning errors into erasures with little or no inner error detection. Complexity, with almost any code, is only moderately greater than for the erasure channel. If error detection of possibly less than one bit per packet is built into the data, it can select the correct decision from a small number of alternatives, and avoid the performance degradation compared to the erasure channel.

Low density codes are generally analyzed in terms of bipartite graphs. These graphs provide important visualizations of the relations between verification sets. The graph also can be described as a binary matrix. For the method developed here, the matrix representation is found more useful. This representation shows that the method is not restricted to low density codes.

Following is a description of how the encoding is done in [13]. This description differs slightly from that in [13], but it is the same basic idea. Each symbol is a packet or multi-bit vector. Assume all packets are the same size:  $r$  bits. Say we have  $k$  data vectors and  $n$  code vectors. A parity check code can be constructed from a  $k \times n$  binary generator matrix  $G$  for a binary  $(n, k)$  code, where

$$[\mathbf{c}'_1 \ \mathbf{c}'_2 \ \dots \ \mathbf{c}'_k] = [\mathbf{d}_1 \ \mathbf{d}_2 \ \dots \ \mathbf{d}_k] G \quad (1)$$

Each  $\mathbf{c}'_i$  and  $\mathbf{d}_i$  is an  $r$ -bit vector, and operations with  $G$  are vector XOR

Add a pseudo-random vector  $\mathbf{t}_i$  to code symbol  $\mathbf{c}'_i$ , so that  $\mathbf{c}_i = \mathbf{c}'_i + \mathbf{t}_i$

$$[\mathbf{c}_1 \ \mathbf{c}_2 \ \dots \ \mathbf{c}_n] = [\mathbf{d}] G + [\mathbf{t}_1 \ \mathbf{t}_2 \ \dots \ \mathbf{t}_n] \quad (2)$$

Since the  $\mathbf{t}_i$  are preselected, if  $r > n$ , as will be assumed in the basic method, it is easy to select them so they are linearly independent. The values  $\mathbf{t}_1 \ \mathbf{t}_2 \ \dots \ \mathbf{t}_n$  are known by the decoder Then

$$[\mathbf{s}_1 \ \mathbf{s}_2 \ \dots \ \mathbf{s}_{n-k}] = [\mathbf{c}_1 \ \mathbf{c}_2 \ \dots \ \mathbf{c}_n] H^T = [\mathbf{t}_1 \ \mathbf{t}_2 \ \dots \ \mathbf{t}_n] H^T \quad (3)$$

The  $\{\mathbf{s}_i\}$  are linearly independent because the rows of  $H$  are linearly independent and  $\{\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_n\}$  are linearly independent. The  $\{\mathbf{s}_i\}$  values also are known by the decoder

Equation (3) is used to determine  $[\mathbf{c}_1 \ \mathbf{c}_2 \ \dots \ \mathbf{c}_n]$ . Then the true code symbols are computed from  $\mathbf{c}'_i = \mathbf{c}_i + \mathbf{t}_i$ . (Only the data symbols actually need to be computed).

## II. THE BASIC METHOD.

The basic method is the most general form. A modification that provides further simplification, particularly for convolutional codes, will be described later.

**Step 1.** Take the (approximately)  $n$  packets that have arrived as row vectors. The number of received packets may differ slightly from  $n$  due to packet deletions, foreign packet insertions, or

multiple receptions of a packet. Assume the packet size  $r$  in bits is such that  $r - (2n-k)$  is sufficiently large (about 16 to 32 bits) to have confidence about no false indications. A modified method could use smaller values of  $r$ . Form a matrix with the received packets as the first set of rows, and then write  $n-k$  rows consisting of the corresponding  $r$  bits of the  $\{s_i\}$ .

**Step 2.** Do elementary column operations on  $r$  columns of the matrix to diagonalize the packet rows. The rows almost always will be linearly independent. Once a point is reached where  $j$  rows are diagonalized and correct values for all members of a check sum  $s_i$  are included, the row that had  $s_i$  will be recognized immediately as a row among the  $n-k$  that is zero in all of the  $r-j$  remaining column positions. As a further reinforcement, the number of 1's in the transformed  $s_i$  must equal the number of 1's in its check equation. The 1's in row  $i$  of  $H$  define the members associated with  $s_i$ . For example, if  $n=12$  and row 1 of  $H$  is 100011000000, suppose  $s_1$  is transformed into 10110.....0. If order is preserved, this means that packets  $c_1$ ,  $c_5$ , and  $c_6$  were correctly received, in that order, but there must be some deletions between  $c_1$  and  $c_5$ . If order is not preserved, it means still that packets  $c_1$ ,  $c_5$ , and  $c_6$  were correctly received and we know their three values, but we don't know yet which packet has which value.

**Step 3.** In many cases sufficient verifications cannot be made solely by looking at the  $\{s_i\}$  individually. A further step automates discovery of any verifying combination sums of  $\{s_i\}$  members. Step 3 will be explained in the Section III examples.

**Step 4.** Determine whether the prior steps have provided sufficient verifications of correct values and positions of packets. If so, solve for the values of the missing packets and extract the packet data. This step is similar to erasure-filling methods.

In [13] with deletions but even with ordering, it was found necessary to try various sums to find a match, since with deletions it is not sure where the verification set members will appear. In the proposed method this trial is not needed, since all existing verification sets are revealed automatically in the first three steps. Thus, the proposed method finds any verifiable sets in a manner simpler than described in [13]. Furthermore, whereas there are only  $n-k$  verification sets in [13], the proposed method, with Step 3, quickly reveals any of the  $2^{n-k}$  possible verifiable sets from the row space of  $H$ . As a result, the proposed method is not restricted to low density codes. It applies to any parity check code. Codes of better distance profile yield better performance. Use of low density codes could provide some simplification. However, even if  $H$  is not low density, many of the row space members of  $H$  are low density.

If order is not preserved, the verified sets still are found without trials in the proposed method, but at this point ambiguity remains as to the order within the set. This ambiguity can be resolved in most cases from other verified sets that have common members. For example, if a particular position is the only one common to two verified sets, that position is found uniquely.

The following cases will be considered:

Case 1. Packets of a code word arrive in order, but deletions, errors, insertions of foreign packets may occur.

Case 2. Packets of code word can arrive out of order. Multiple receptions of a packet may occur. a) All arrive with no deletions or errors.

b) Deletions, insertions of foreign packets, packet errors.

### III. EXAMPLES OF CASE 1.

In this case packets arrive in order, with possible deletions. In the examples, consider the following particular (10,3) code.

$$G = \begin{pmatrix} 11 & 01 & 11 & 00 & 00 \\ 00 & 11 & 01 & 11 & 00 \\ 00 & 00 & 11 & 01 & 11 \end{pmatrix} \quad (4)$$

This code has a minimum Hamming distance of 5.

Parity check matrix and associated verification sets.

$$H = \begin{pmatrix} 11 & 00 & 00 & 00 & 00 \\ 10 & 11 & 00 & 00 & 00 \\ 11 & 10 & 11 & 00 & 00 \\ 00 & 11 & 10 & 11 & 00 \\ 00 & 00 & 11 & 10 & 11 \\ 00 & 00 & 00 & 11 & 10 \\ 00 & 00 & 00 & 00 & 11 \end{pmatrix} \quad (5)$$

$$\begin{aligned} s_1: & (c_1, c_2) \\ s_2: & \{c_1, c_3, c_4\} \\ s_3: & (c_1, c_2, c_3, c_5, c_6) \\ s_4: & (c_3, c_4, c_5, c_7, c_8) \\ s_5: & \{c_5, c_6, c_7, c_9, c_{10}\} \\ s_6: & (c_7, c_8, c_9) \\ s_7: & (c_9, c_{10}) \end{aligned}$$

$c_1$	1000000000.....0
$c_2$	0100000000.....0
$c_3$	0010000000.....0
$c_4$	0001000000.....0
$c_5$	0000100000.....0
$c_6$	0000010000.....0
$c_7$	00000010000.....0
$c_8$	00000001000.....0
$c_9$	00000000100.....0
$c_{10}$	00000000010.....0
$s_1$	1100000000.....0
$s_2$	1011000000.....0
$s_3$	1110110000.....0
$s_4$	00111011000.....0
$s_5$	00001110110.....0
$s_6$	00000001110.....0
$s_7$	00000000110.....0

Figure 1. Ordered case with no deletions.

The result after column operations with no deletions is shown in Figure 1. Note that in this ordered, error-free case, the last 7 rows after column operation are in the space, and match exactly the 7 rows of H.

All the sets are verified.

A. Finding deletions, ordering known

After deletions or errors, without sequence numbers the positions of the deletion/error packets may not be known. With the large value of  $r$  used, an error is almost certain to have the same effect as a deletion. However, Section VIII makes some qualifications about this statement.

It may be possible to identify some verification sets. For example, if the set 1,3,4 is verified directly, the exact positions of these three packets as well as their values are known if ordered reception is known to exist. From other verifications, it often is possible to verify additional packets, and solve for unverified deleted packet values. In the example code, say  $c_2$ ,  $c_5$  and  $c_6$  are deleted (marked with x by the matrix H.) . Then  $s_2: \{c_1, c_3, c_4\}$ ,  $s_6: (c_7, c_8, c_9)$ ,  $s_7: (c_9, c_{10})$  are verified from the elementary column operations (see Figure 2), and values  $c_1, c_3, c_4, c_7, c_8, c_9$ , and  $c_{10}$  are all known in value and position.

	<b>x</b>	<b>xx</b>					
H =	11 00 00 00 00		x	$s_1: (c_1, c_2)$	$c_1$	1000000000.....0	
	10 11 00 00 00			$s_2: \{c_1, c_3, c_4\}$	$c_3$	0100000000.....0	
	11 10 11 00 00		x	$s_3: (c_1, c_2, c_3, c_5, c_6)$	$c_4$	0010000000.....0	
	00 11 10 11 00		x	$s_4: (c_3, c_4, c_5, c_7, c_8)$	$c_7$	0001000000.....0	
	00 00 11 10 11		x	$s_5: \{c_5, c_6, c_7, c_9, c_{10}\}$	$c_8$	0000100000.....0	
	00 00 00 11 10			$s_6: (c_7, c_8, c_9)$	$c_9$	0000010000.....0	
	00 00 00 00 11			$s_7: (c_9, c_{10})$	$c_{10}$	00000010000.....0	
					$s_1$	xxxxxxxxxxxxxxxxxxxx	
					$s_2$	1110000000.....0	
					$s_3$	xxxxxxxxxxxxxxxxxxxx	
					$s_4$	xxxxxxxxxxxxxxxxxxxx	
					$s_5$	xxxxxxxxxxxxxxxxxxxx	
					$s_6$	0001110000.....0	
					$s_7$	00000110000.....0	

$s_1$  is a known value. From  $s_1 = c_1 + c_2$  we solve for  $c_2$ .  
 From  $s_4 = c_3 + c_4 + c_5 + c_7 + c_8$  we find  $c_5$ . Then we can find  $c_6$  from  $s_3$  or  $s_5$ . Now we know all packets and the order.

But suppose the deleted packets were instead  $c_1, c_3, c_9$ . In this case there will be no verification with any of the individual  $\{s_i\}$ . By inspection we could find independent verifiable sets:

$$\begin{aligned}
 s_2 + s_3 &= c_2 + c_4 + c_5 + c_6 \\
 s_6 + s_7 &= c_7 + c_8 + c_{10} \\
 s_5 + s_7 &= c_5 + c_6 + c_7 \\
 s_1 + s_2 + s_4 &= c_2 + c_5 + c_7 + c_8 .
 \end{aligned}
 \tag{6}$$

Just the first two of these are sufficient to verify all seven correct receptions. Since the code minimum distance is 5, simple erasure-filling decoding or the  $\{s_i\}$  could solve for  $c_1, c_3, c_9$ .

Figure 2. Ordered receptions with deletions.

The finding of these verification combinations can be automated by further array column operations. This is **step 3** of the basic method.

**Step 3.** Take the unverified set (all 7 in this case) and attempt to diagonalize this set with column operations only beyond the ones diagonalized with the received packets (it is for this reason that we require  $r > 2n-k$  rather than just  $r > n$ ). This will yield Figure 3, which shows how to find the verifications. For the pair case, such as two pairs illustrated in Figure 3, it could also be observed even without the operations that the transformed  $s_2$  and  $s_3$  were identical beyond the initial 7 columns (also  $s_5$  and  $s_6$ ,  $s_6$  and  $s_7$ ).

The additional diagonalizations reveal four dependent sets in the space. Translate the dependencies into a 4 by 7 matrix. Place a "1" in the four rows in the dependent positions 3,4,6,7, respectively. Then add underlined 1's showing the relation to independent syndromes 1, 2 and 5. The right side of Figure 3 shows the matrix multiplication that reveals the verification sets.

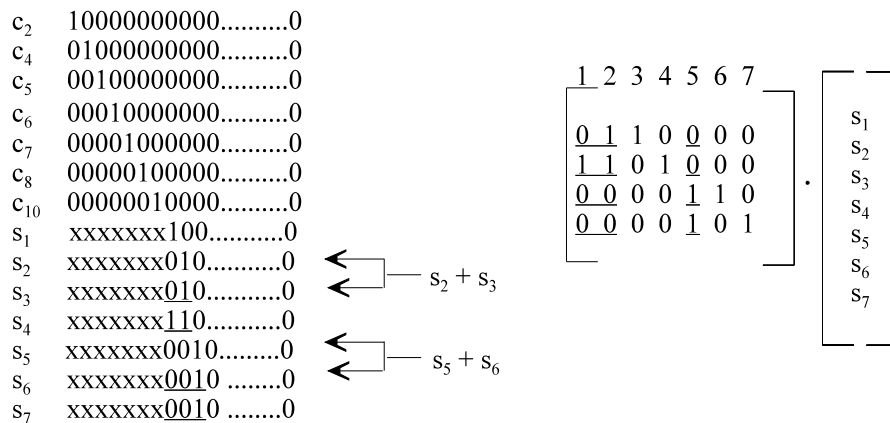


Figure 3. Step 3. Automated discovery of additional verification sets.

These discovered independent verification sets, related to the row space of H, are:

- $s_2 + s_3$  : 0101110000
- $s_1 + s_2 + s_4$ : 0100101100
- $s_5 + s_6$  : 0000110101
- $s_5 + s_7$  : 0000111000

Actually, it only is necessary to find  $k$  data symbols. There are 3 data symbols in the example. If 3 symbols are verified that correspond to linearly independent columns of  $G$ , all other symbols can be computed. Thus, in the case of errors in positions 1,3,9, verification of just the one set  $\mathbf{s}_2 + \mathbf{s}_3 = \mathbf{c}_2 + \mathbf{c}_4 + \mathbf{c}_5 + \mathbf{c}_6$  is sufficient, since columns 2,4,5,6 of  $G$  are rank 3. Thus a pattern of 6 deletions, in positions 1,3,7,8,9,10, can be solved in the ordered case.

For the case of erasures where packets have sequence numbers and error detection, erasure-filling can solve many cases of seven deletions in this code. However, seven deletions cannot be solved by the verification method. This can be seen easily by looking at the code in systematic form:

$G = [I_3 \ P]$ ;  $H = [P^T \ I_7]$ . If just the first 3 symbols are received correctly and this is known, the code is decoded. But if we don't know where the deletions are, it can be seen from the systematic form of  $H$  that the row space of  $H$  does not contain a check vector  $[xxx0000000]$  where any of the  $x$ 's are nonzero, so we can't verify any correct location. Similar things could be said about any set of three linearly independent positions, since those positions can be diagonalized and columns permuted into a systematic form. So three ( $k$  in general) known symbols will either be independent, and then we can't find a verification, or dependent, where we might get a verification but not enough information to solve for the data values. For example,  $\mathbf{s}_2: \{\mathbf{c}_1, \mathbf{c}_3, \mathbf{c}_4\}$  gives a verification of three values, but not three independent equations to solve for the data because columns 1,3, and 4 of  $G$  are not linearly independent.

With a minimum distance of 5 it is possible to fill in any four erasures. In the case of four deletions there may be additional uncertainty as to which four are erased. This creates an ambiguity problem if the deleted columns erase 4 of the positions of a weight 5 code word. In this case, from  $G$ , one code word is positions 1,2,4,5,6. Suppose we erase positions 1,2,5,6.

Another independent verification can be found by Step 3. Or, an analyst who knows where the errors are can see the result by inspection of columns 1,2,5,6 of  $H$ .

xx	xx			1 2 5 6
11 00 00 00 00	x	$\mathbf{s}_1: (\mathbf{c}_1, \mathbf{c}_2)$		1 1 0 0
10 11 00 00 00	x	$\mathbf{s}_2: \{\mathbf{c}_1, \mathbf{c}_3, \mathbf{c}_4\}$		1 0 1 1
11 10 11 00 00	x	$\mathbf{s}_3: (\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_5, \mathbf{c}_6)$		1 1 1 1
00 11 10 11 00	x	$\mathbf{s}_4: (\mathbf{c}_3, \mathbf{c}_4, \mathbf{c}_5, \mathbf{c}_7, \mathbf{c}_8)$		0 0 1 0
00 00 11 10 11	x	$\mathbf{s}_5: \{\mathbf{c}_5, \mathbf{c}_6, \mathbf{c}_7, \mathbf{c}_9, \mathbf{c}_{10}\}$		0 0 1 1
00 00 00 11 10		$\mathbf{s}_6: (\mathbf{c}_7, \mathbf{c}_8, \mathbf{c}_9)$		0 0 0 0
00 00 00 00 11		$\mathbf{s}_7: (\mathbf{c}_9, \mathbf{c}_{10})$		0 0 0 0

Note that rows 1,3,5 add to zero. Thus we see that Step 3 of the basic method will reveal:

$$\mathbf{s}_1 + \mathbf{s}_3 + \mathbf{s}_5 = \mathbf{c}_3 + \mathbf{c}_7 + \mathbf{c}_9 + \mathbf{c}_{10}. \quad (7)$$

The deletions are narrowed to 5 positions, but we don't know which, if any, position is right. We have one more reception, but don't know if it is right or which of the 5 it is. Given a value for one of the positions, the columns of H show four independent equations that could be solved for the other four. Five different results exist consistent with the syndromes, so the right combination can't be found without outside information.

The conclusion is that the verification method can find deletions up to where all code words have at least two positions undeleted, whereas erasure filling requires only that all code words have at least one position undeleted. This does not preclude correcting some cases of 4 or more deletions for this code - we have already seen examples of six deletions that can be solved for the deletions.

#### IV. EXAMPLES OF CASE 2

##### A. Case 2a. Packets out of order but no deletions.

The array after transformation with out-of-order reception is shown in Figure 4.

Each of  $\mathbf{s}_1$  to  $\mathbf{s}_7$  verifies its group and translates into the 7 groups listed above. For example, the two "1"s in the row for  $\mathbf{s}_1$  must be for  $\mathbf{c}_1$  and  $\mathbf{c}_2$ , but it is not yet known which is which. The bottom 7 rows are just the H matrix with its columns permuted by the mis-ordering. Since the code minimum distance is greater than 2, each column of H is different. So the proper placement of a packet is the column position in the actual H. Thus ordering is not a problem if all packet values are verified.

$\mathbf{c}_6$	1000000000.....0
$\mathbf{c}_3$	0100000000.....0
$\mathbf{c}_1$	0010000000.....0
$\mathbf{c}_7$	0001000000.....0
$\mathbf{c}_5$	0000100000.....0
$\mathbf{c}_4$	0000010000.....0
$\mathbf{c}_{10}$	0000001000.....0
$\mathbf{c}_8$	0000000100.....0
$\mathbf{c}_9$	0000000010.....0
$\mathbf{c}_2$	0000000001.....0
$\mathbf{s}_1$	00100000010.....0
$\mathbf{s}_2$	0110010000.....0
$\mathbf{s}_3$	11101000010.....0
$\mathbf{s}_4$	0101110100.....0
$\mathbf{s}_5$	10011010100.....0
$\mathbf{s}_6$	00010001100.....0
$\mathbf{s}_7$	00000100100.....0

Figure 4. Out-of-order reception with no deletions.

Case 2b. Finding deletions, ordering unknown.

For the same code in prior examples, consider the case where  $c_2$  and  $c_5$  are deleted.

Besides  $s_2$ ,  $s_6$  and  $s_7$ , there are two other independent verifications. The decoder could obtain them by Step 3 of the basic method. The analyst can deduce the combinations from columns 2 and 5 of H:

$$\begin{array}{rcccccc}
 & \mathbf{x} & \mathbf{x} & & & & 2 & 5 \\
 & 11 & 00 & 00 & 00 & 00 & \mathbf{x} & 1 & 0 \\
 & 10 & 11 & 00 & 00 & 00 & & \mathbf{s}_2: \{c_1, c_3, c_4\} & 0 & 0 \\
 \mathbf{H} = & 11 & 10 & 11 & 00 & 00 & \mathbf{x} & & 1 & 1 \\
 & 00 & 11 & 10 & 11 & 00 & \mathbf{x} & & 0 & 1 \\
 & 00 & 00 & 11 & 10 & 11 & \mathbf{x} & & 0 & 1 \\
 & 00 & 00 & 00 & 11 & 10 & & \mathbf{s}_6: (c_7, c_8, c_9) & 0 & 0 \\
 & 00 & 00 & 00 & 00 & 11 & & \mathbf{s}_7: (c_9, c_{10}) & 0 & 0
 \end{array}$$

The dependencies are: the three zero rows 2,6,7, the sum of rows 4 and 5, and the sum of rows 1,3 and 4.

$$s_4 + s_5 = c_3 + c_4 + c_6 + c_8 + c_9 + c_{10}.$$

$$s_1 + s_3 + s_4 = c_4 + c_6 + c_7 + c_8$$

All correct values are in the verifications. Actually, there are five independent members of the row space of H, with a zero column in the two error positions. These are actually the verification vectors. Since the minimum distance of the code is 5, the minimum distance deleting two columns must be at least  $5 - 2 = 3$ . Thus every column of a matrix of the 5 verification vectors, excluding 2 and 5, is different:

$$\begin{array}{c}
 1011000000 \\
 0000001110 \\
 0000000011 \\
 0011010111 \\
 0001011100
 \end{array}$$

Thus each of the eight verified values can be exactly placed by its column position in the verifications. It is then a simple matter to solve for the two “erasures” from the 8 known values.

Let’s look at three deletions, all part of a code word of weight 5.  $c_2$ ,  $c_5$  and  $c_6$  are missing. It will be seen that this case does not lead to a unique solution.

Look at columns 2,5,6 of H:

100
000
111
010
011
000
000

Now there are four independent members of the row space of H with zeroes in the missing positions: rows 2, 6,7, (found in Step 2) and row 1 + row 3 + row 5 (found in Step 3). These correspond to the four verification sets:

```
10 11 00 00 00
00 00 00 11 10
00 00 00 00 11
00 10 00 10 11
```

There is one pairing of columns: 1 and 4. The places and values of positions 3, 7,8,9,10 are known. We need to find 3 unknown values, and establish which of two values is  $\mathbf{c}_1$  and which is  $\mathbf{c}_4$ .

The positions 1,2,4,5,6 correspond to the 1's in the binary code word defined by (4) and (5). As in case 1 with four errors out of 5, it is not possible to uniquely determine which placement of the two values  $\mathbf{c}_1$  and  $\mathbf{c}_4$  is correct without outside information, such as internal error detection. Say the right pairing is  $\mathbf{c}_1 = \mathbf{a}$ ,  $\mathbf{c}_4 = \mathbf{b}$ . Since there are three unknowns and a hamming distance of 5, the values of  $\mathbf{c}_2$ ,  $\mathbf{c}_5$  and  $\mathbf{c}_6$  could be determined correctly, just by means of filling in three erasures. The result in positions 1,2,4,5,6 is  $(\mathbf{a}, \mathbf{c}_2, \mathbf{b}, \mathbf{c}_5, \mathbf{c}_6)$ . Suppose we selected

$$\mathbf{c}_1 = \mathbf{b}, \mathbf{c}_4 = \mathbf{a}.$$

From  $\mathbf{s}_1 = \mathbf{c}_1 + \mathbf{c}_2$ ,

$\mathbf{c}_2$  is altered by  $\mathbf{b} + \mathbf{a}$ . From

$$\mathbf{s}_4 = \mathbf{c}_3 + \mathbf{c}_4 + \mathbf{c}_5 + \mathbf{c}_7 + \mathbf{c}_8, \quad \mathbf{c}_5 \text{ is altered by } \mathbf{b} + \mathbf{a}.$$

Also, from  $\mathbf{s}_5$ ,  $\mathbf{c}_6$  is computed from  $\mathbf{c}_5$  and also is altered by  $\mathbf{b} + \mathbf{a}$ . The result is  $(\mathbf{b}, \mathbf{c}_2 + \mathbf{b} + \mathbf{a}, \mathbf{a}, \mathbf{c}_5 + \mathbf{a}, \mathbf{c}_6 + \mathbf{b} + \mathbf{a})$ . The difference is  $(\mathbf{b} + \mathbf{a}, \mathbf{b} + \mathbf{a}, \mathbf{b} + \mathbf{a}, \mathbf{b} + \mathbf{a}, \mathbf{b} + \mathbf{a})$ , which adds to zero in the syndrome calculation. The two results show no syndrome difference.

## V. GENERALIZATIONS

### *A. Out-of-order receptions*

Given linearly independent received packets, with out-of-order receptions and no sequence numbers, all deletions whose patterns do not cover all but two positions of any code word can be decoded. With a minimum distance of  $d$ , all cases of  $d-3$  or less deletions can be decoded, plus many cases of greater than  $d-3$  deletions but satisfying the coverage constraint. Consider the check equations  $[S] = [s_1 s_2 \dots s_{n-k}]^T = H [c_1 c_2 \dots c_n]^T$  (9)

Assume without loss of generality that all the deletions are in the first  $e$  positions and the error set satisfies the coverage constraint. These  $e$  columns must be linearly independent since they don't cover a code word. Then the  $H$  matrix can be transformed by row operations into the form:

$$\text{Transf.}[S] = \left[ \begin{array}{c|c} I_e & A \\ \hline 0 & B \end{array} \right] \cdot [c_1 c_2 \dots c_n]^T \quad (10)$$

where  $\text{Transf.}[S]$  in (10) is a transform of  $S$  with the same row operations as  $H$ .

The submatrix  $B$  can be created from the derivation of  $n-k-e$  verification sets derived as described in Steps 2 and 3. The decoder knows which combinations of rows of  $H$  have gone into each combination set, which are exactly the ones that would be zero in all the deleted positions. Thus the matrix of verified sets is just  $B$  with columns permuted. All columns of  $B$  are different and nonzero, since no code word has been reduced to less than 3 nonzero terms by the deletions. Thus every column of  $B$  is different, so the verification equations can find the correct column permutation into  $B$ . The other  $e$  row equations can be used to solve for the deleted code words in the now known deleted positions.

If a code word were reduced to a weight two, then  $B$  would have two identical columns, and there would be a pair placement ambiguity. There would then be no unique solution.

### *B. Ordered receptions.*

The same submatrix  $B$  as in equation (10) is discovered by Steps 1-3 of the basic method. With ordered reception and deletions and no sequence numbers, the case of covering all but two positions of a code word will result in two columns of  $B$  being identical, but there can be no zero columns in  $B$ . Thus every non-deleted position will be found in some verification set, and the deleted positions will be identified as the unverified positions. But in this case the order within each verification set will always be known. Since two columns of  $B$  are identical, the pair of possibly ambiguous positions will appear in the same verification set. Since the order within the verification set is known, the two positions can be placed uniquely. The deletions can then be found by erasure-filling methods. With a minimum distance of  $d$ , all cases of  $d-2$  or less deletions can be decoded, plus many cases of greater than  $d-2$  deletions if they satisfy the code word coverage constraint.

### *C. Use of a systematic code*

It is possible to use the method with  $H$  in systematic form. Assume  $G = [ I_k \ P ]$ ;  
 $H = [ P^T \ I_{n-k} ]$ , data first, transmission order left to right. This would make it easier to extract the data packets. Also, it is easy to see when all the data packets are verified, at which point decoding need go no further. However, it is not the best form for the idea of partitioning (see section VII ) to reduce the verification search computation. This is because no verifications can be made until at least the  $(k+1)$ th packet reception.

## VI. CONSIDERATIONS OF LINEAR DEPENDENCE

### *A. General considerations*

The linearly independence of components  $\{t_1, t_2, \dots, t_n\}$  is the main factor making linear dependence among different packets unlikely. The components  $\{c_1, c_2, \dots, c_n\}$  are dependent due to code relation (1). But the  $\{c_1, c_2, \dots, c_n\}$  are almost surely linearly independent, due to the large value of  $b$  and the components  $\{t_1, t_2, \dots, t_n\}$ . Also, foreign packets or packets with error are very likely to be linearly independent.

In the second row of the verification, the probability of a dependence is the probability the first two are equal, about  $2^{-r}$  for a random difference. If two correct copies of a packet happen to be received, this will show up in the column operations. The duplicate packet event is handled in the unordered case simply by ignoring one of the two receptions.

After  $j$  receptions that are independent, there is a space of  $2^j$  combinations, so the probability the next packet will be independent is about  $2^{-(r-j)}$ . This probability doubles for each new reception. A dependency usually occurs near the end of the diagonalization, at which time most verifications have already been made. Step 3 looks for additional verifying combinations of the  $n-k$  syndromes using the remaining  $r-n$  bit positions. Thus  $r - (n + (n-k)) = r - (2n-k)$  should be about 32 bits for negligible chance of linear dependence, though smaller values may be acceptable.

In the elementary column operations any dependency will be revealed, and the number of received packets in the dependency set is visible. The easiest thing to do is to discard the packet first discovered as dependent. Since the probability of deletion usually is much larger than the probability of a linear dependence, deleting a dependent reception has little effect on performance.

It is possible to do better than discarding the last packet of a dependency. Note that any one of the members could be the one discarded and the rest would be independent. But some deletions are less harmful than others. If an additional deletion brings the deletion set too close to covering a code word, it could cause a decoder failure, but if a different packet were deleted, it might not have this affect. Unfortunately, the critical packets may not be known until a later stage in the decoding, so deleting some less critical packet may require additional work, such as retries. Possibly in some cases there might be enough information from prior verifications to make the decision which to delete. To delete any but the most recent requires at least some additional column operations.

B. Example of three deletions and a dependency for the same (10,3) code, case 1)

Reception is ordered,  $c_2$ ,  $c_4$  and  $c_5$  are deleted, and  $c_6$  is part of an order 3 dependency:

$$c_1 + c_3 + c_6 = 0$$

Only two syndromes,  $s_6$  and  $s_7$ , appear directly in

Figure 5. They evaluate positions 7,8,9,10. Step 3

diagonalization produces

$s_1 + s_3 + s_5$ , which should be  $c_3 + c_7 + c_9 + c_{10}$ .

$s_1 + s_2 + s_3 + s_4$ , which should be  $c_1 + c_3 + c_6 + c_7 + c_8$ .

The verification matrix from these discoveries is:

0000001110

0000000011

0010001011

1010001100

$s_6$ ,  $s_7$  and  $s_1 + s_3 + s_5$  (first 3 rows of the verification matrix) are consistent with this.

3 7 8 9 10

0 0 1 1 1 0

0 0 0 0 1 1

0 1 1 0 1 1

The  $s_1 + s_2 + s_3 + s_4$  component does not show 5

components corresponding to  $c_1 + c_3 + c_6 + c_7 + c_8$ . It shows

0 0 1 1 0 0, which has the wrong number of entries,

But the two "1" places correspond to verified  $c_7$  and  $c_8$ , the order three dependency almost surely is

$c_1 + c_3 + c_6 = 0$ , since it would make  $c_1 + c_3 + c_6 + c_7 + c_8 = c_7 + c_8$ . This can be considered as verification for positions 1,3,6 as well as 7,8,9,10.

Now: 1)  $c_4$  can be computed from  $s_2$  knowing  $c_1$ ,  $c_3$ ;

2)  $c_2$  can be computed from  $s_1$ ;

3)  $c_5$  can be computed from  $s_3$ .

If  $c_6$  had been treated as a deletion, the deletion of all positions 2,4,5,6 that are one from the code word 1,2,4,5,6 would have failed to find a unique result. So using the dependence is better than calling it a deletion.

$c_1$	1000000000.....0 d
$c_3$	0100000000.....0 d
$c_6$	1100000000.....0 d
$c_7$	0010000000.....0
$c_8$	0001000000.....0
$c_9$	0000100000.....0
$c_{10}$	0000010000.....0
$s_1$	xxxxxx1000.....0
$s_2$	xxxxxx0100.....0
$s_3$	xxxxxx0010.....0
$s_4$	xxxxxx1110.....0
$s_5$	xxxxxx1010.....0
$s_6$	0011100000.....0
$s_7$	0000110000.....0
$s_1 + s_3$	
+ $s_5$	0110110000.....0
$s_1 + s_2$	
+ $s_3 + s_4$	0011000000.....0?

Figure 5. Three deletions and a dependency, ordered case.

*C. Same as B, but change the deletion.*

Although we might not know what to delete, suppose we delete  $c_3$  to remove the dependence, and redo steps 1-3. Note that deleting  $c_6$  or  $c_1$  would have failed due to too close to a code word.

In this case 1,6,7,8,9,10 are verified, and there are four independent equations to solve for the four unknowns.

So deleting  $c_3$  works because it is not needed to obey the code word coverage constraint.

$c_1$	1000000000.....0
$c_6$	0100000000.....0
$c_7$	0010000000.....0
$c_8$	0001000000.....0
$c_9$	0000100000.....0
$c_{10}$	0000010000.....0
$s_1$	xxxxxx1000.....0
$s_2$	xxxxxx0100.....0
$s_3$	xxxxxx0010.....0
$s_4$	xxxxxx0001.....0
$s_5$	xxxxxx0101.....0
$s_6$	0011100000.....0
$s_7$	0000110000.....0
$s_2+s_4$	
+ $s_5$	1110110000.....0

Figure 6. Three deletions and dependency  $c_3$  deleted, ordered case.

D. A dependency which includes an already verified packet.

Suppose packets  $\mathbf{c}_6$ ,  $\mathbf{c}_7$  and  $\mathbf{c}_8$  are deleted, and there is a dependency  $\mathbf{c}_1 + \mathbf{c}_5 + \mathbf{c}_{10} = 0$ . The array would appear as in Figure 7. But  $\mathbf{s}_1$  and  $\mathbf{s}_2$  already verify and determine the first four position values after just the first 4 receptions.

The diagonalization could be halted after  $\mathbf{c}_4$ , and restarted with the known values subtracted:

$$\begin{aligned} \mathbf{s}_3' &= \mathbf{s}_3 + \mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 \\ \mathbf{s}_4' &= \mathbf{s}_4 + \mathbf{c}_3 + \mathbf{c}_4 \\ \mathbf{s}_5' &= \mathbf{s}_5 \\ \mathbf{s}_6' &= \mathbf{s}_6 \\ \mathbf{s}_7' &= \mathbf{s}_7 \end{aligned}$$

There is no longer any dependency showing.  $\mathbf{s}_4' + \mathbf{s}_6' = \mathbf{c}_5 + \mathbf{c}_9$ , and  $\mathbf{s}_7' = \mathbf{c}_9 + \mathbf{c}_9$  verify positions 5,9,10, leaving positions 6,7,8 that can be found as erasure filling.

Even without dependency discovery, the idea of partitioning the verification search is useful. This partitioning approach will be discussed further in Section VII.

$\mathbf{c}_1$	1000000000.....0		
$\mathbf{c}_2$	0100000000.....0	$\mathbf{c}_5$	1000000000.....0
$\mathbf{c}_3$	0010000000.....0		
$\mathbf{c}_4$	0001000000.....0	$\mathbf{c}_9$	01000000100.....0
$\mathbf{c}_5$	0000100000.....0	$\mathbf{c}_{10}$	00100000000.....0
		$\mathbf{s}_3'$	xxx10000.....0
$\mathbf{c}_9$	00000000100.....0	$\mathbf{s}_4'$	xxx01000.....0
$\mathbf{c}_{10}$	10001000000.....0	$\mathbf{s}_5'$	xxx00100.....0
$\mathbf{s}_1$	11000000000.....0	$\mathbf{s}_6'$	xxx01000.....0
$\mathbf{s}_2$	10110000000.....0	$\mathbf{s}_7'$	011000000.....0
$\mathbf{s}_3$	xxxxxxxxxx.....0		
$\mathbf{s}_4$	xxxxxxxxxx.....0		
$\mathbf{s}_5$	xxxxxxxxxx.....0		
$\mathbf{s}_6$	xxxxxxxxxx.....0		
$\mathbf{s}_7$	100010001000.....0		

a) Array showing dependency

b) Array redone with known packets subtracted.

Figure 7. Ordered case with 3 deletions and a dependency with a verified term.

E. Case 2b, two deletions plus one dependency

Since order is arbitrary, this case is restricted to not covering at least 3 positions of any code word. Suppose  $c_2$  and  $c_5$  are deleted, and  $c_8+c_1+c_4+c_7 = 0$ . Let the order be 3,1,6,4,8,7,9,10. Figure 8a shows the column operation result, including additional verification sets from Step 3.

Some results (indicated by “?”) show an inconsistency due to use of the dependent  $c_7$  entry.  $s_6$  clearly is incorrect:  $s_2$  and  $s_6$  cannot have a common entry.  $s_1+s_3+s_4$  has the wrong number of ones. The real contributions can be found easily. For Figure 8b, introduce a new position (italic) to mark the dependent  $c_7$  (we don't need to know the position is really 7). Add the dependency pattern 01011 1 to the old  $c_7$  row and to the ? rows. Now  $s_6$  and  $s_1+s_3+s_4$  are correctly verified as their having two common values and proper weights are consistent.

Write down the five verification sets for positions 1,3,4,6,7,8,9,10:

	Actual order 3,1,6,4,8,7,9,10	order 1,3,4,6,7,8,9,10	From H matrix, italics for deleted
$s_2$	11010 0 00	11100000	10 11 00 00 00
$s_6$	00011 <i>1</i> 00	00001110	00 00 00 11 10
$s_7$	00000 0 11	00000011	00 00 00 00 11
$s_4+s_5$	10111 0 1 1	01110111	00 11 01 01 11
$s_1+s_3+s_4$	00111 <i>1</i> 00	00111100	00 01 01 11 00

Note the eight columns are all different. Placement can be determined by comparison with the columns of the independent H matrix row vectors. Then erasure-filling finds the deleted packets.

$c_3$	1000000000.....0	$c_3$	10000 00000.....0
$c_1$	0100000000.....0 d	$c_1$	01000 00000.....0 d
$c_6$	0010000000.....0	$c_6$	00100 00000.....0
$c_4$	0001000000.....0 d	$c_4$	00010 00000.....0 d
$c_8$	0000100000.....0 d	$c_8$	00001 00000.....0 d
$c_7$	0101100000.....0 d	$c_7$	00000 <i>1</i> 00000.....0 d
$c_9$	0000010000.....0	$c_9$	00000 10000.....0
$c_{10}$	0000001000.....0	$c_{10}$	00000 01000.....0
$s_1$	xxxxxxxxxxxxxxxxxxx		place added for dependent term
$s_2$	1101000000.....0		Add dependency to ? sets
$s_3$	xxxxxxxxxxxxxxxxxxx		01011 <i>1</i>
$s_4$	xxxxxxxxxxxxxxxxxxx		
$s_5$	xxxxxxxxxxxxxxxxxxx		
$s_6$	0101010000.....0 ?	$s_6$	00001 <i>1</i> 10000.....0
$s_7$	0000011000.....0		
$s_4+s_5$	1011111000.....0		
$s_1+s_3$		$s_1+s_3$	00111 <i>1</i> 00000.....0
$+s_4$	0110000000.....0 ?	$+s_4$	
	a) Original computation		b) Adjusting $c_7$ for new view.

Figure 8. Out-of-order with two deletions and a dependency.

### *F. Case of a dependent foreign packet*

The only significant effect of an independent foreign packet would be to add another term to the diagonalization. It would be extremely unlikely to appear in any verification set. Also, in most cases a dependent foreign packet would not be a problem. The most troublesome case is where a foreign packet  $\mathbf{f}$  replaces a packet which is included in its dependency set, and the set is the right size to fool a verification set.

In the code example  $\mathbf{s}_4 = \mathbf{c}_3 + \mathbf{c}_4 + \mathbf{c}_5 + \mathbf{c}_7 + \mathbf{c}_8$  is a dependent set, and suppose  $\mathbf{f} = \mathbf{c}_1 + \mathbf{c}_5 + \mathbf{c}_8$  replaces  $\mathbf{c}_8$ . Although  $\mathbf{f}$  is dependent it would appear independent because  $\mathbf{c}_8$  is not actually present. Then  $\mathbf{s}_4 = \mathbf{c}_1 + \mathbf{c}_3 + \mathbf{c}_4 + \mathbf{c}_7 + \mathbf{f}$  would appear to be a legitimate verification set since it has the right number of entries, and  $\mathbf{f}$  would appear to be  $\mathbf{c}_8$ . This would in most cases cause contradictions with other verification sets, but would create extra work or ambiguity for the decoder. Fortunately this event is very rare for  $b - (2n - k) \geq 32$  bits.

## VII. PARTITIONING THE VERIFICATION SEARCH

The verification discovery method as described requires packet size in bits to be about twice as large as number of packets  $n$ , particularly  $r > 2n - k$ . This assumes it is necessary to diagonalize all  $n$  positions and the  $n - k$  syndrome component positions at one time. However, in most cases verifications are discovered in much smaller intervals. In a long code, verification and solution of all packets up to the  $j$ th packet,  $j < n$  might allow restart of the diagonalization at packet  $j + 1$ . Position sets that have not yet been verified can have their verified components subtracted. Convolutional codes are especially well suited for this purpose.

### *A. A Convolutional code example*

Convolutional codes may have very large code length  $n$  until terminated, but verification sets may extend only over a small multiple of the constraint length:  $2(m + 1)$  output symbols for a  $(2, 1, m)$  code. Actually, the example  $(10, 3)$  code is a terminated  $(2, 1, 2)$  convolutional code with

$$G(D) = [1+D^2 \quad | \quad 1+D+D^2]. \quad (11)$$

Let's extend this same code to a terminated convolutional code equivalent to an (20,8) block code. Consider Case 1, with deletions and ordered reception.

$$G = \begin{matrix} 11\ 01\ 11\ 00\ 00\ 00\ 00\ 00\ 00\ 00 \\ 00\ 11\ 01\ 11\ 00\ 00\ 00\ 00\ 00\ 00 \\ 00\ 00\ 11\ 01\ 11\ 00\ 00\ 00\ 00\ 00 \\ 00\ 00\ 00\ 11\ 01\ 11\ 00\ 00\ 00\ 00 \\ 00\ 00\ 00\ 00\ 11\ 01\ 11\ 00\ 00\ 00 \\ 00\ 00\ 00\ 00\ 00\ 11\ 01\ 11\ 00\ 00 \\ 00\ 00\ 00\ 00\ 00\ 00\ 11\ 01\ 11\ 00 \\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 11\ 01\ 11 \end{matrix} \quad (12)$$

$$H = \begin{matrix} 11\ 00\ 00\ 00\ 00\ 00 & | & 00\ 00\ 00\ 00\ 00\ 00 & \mathbf{s}_1 \\ 10\ 11\ 00\ 00\ 00\ 00 & | & 00\ 00\ 00\ 00\ 00\ 00 & \mathbf{s}_2 \\ 11\ 10\ 11\ 00\ 00\ 00 & | & 00\ 00\ 00\ 00\ 00\ 00 & \mathbf{s}_3 \\ 00\ 11\ 10\ 11\ 00\ 00 & | & 00\ 00\ 00\ 00\ 00\ 00 & \mathbf{s}_4 \\ 00\ 00\ 11\ 10\ 11\ 00 & | & 00\ 00\ 00\ 00\ 00\ 00 & \mathbf{s}_5 \\ 00\ 00\ 00\ 11\ 10\ 00 & | & 11\ 00\ 00\ 00\ 00\ 00 & \mathbf{s}_6 \\ \underline{00\ 00\ 00\ 00\ 11\ 00} & | & \underline{10\ 11\ 00\ 00\ 00\ 00} & \mathbf{s}_7 \\ 00\ 00\ 00\ 00\ 00\ 00 & | & 11\ 10\ 11\ 00\ 00\ 00 & \mathbf{s}_8 \\ 00\ 00\ 00\ 00\ 00\ 00 & | & 00\ 11\ 10\ 11\ 00\ 00 & \mathbf{s}_9 \\ 00\ 00\ 00\ 00\ 00\ 00 & | & 00\ 00\ 11\ 10\ 11\ 00 & \mathbf{s}_{10} \\ 00\ 00\ 00\ 00\ 00\ 00 & | & 00\ 00\ 00\ 11\ 10\ 00 & \mathbf{s}_{11} \\ 00\ 00\ 00\ 00\ 00\ 00 & | & 00\ 00\ 00\ 00\ 11\ 00 & \mathbf{s}_{12} \end{matrix} \quad (13)$$

The reason for the horizontal and vertical lines in H will be explained shortly.

At the start, the first pair can be verified just with  $\mathbf{s}_1$ . A single deletion in the first pair can be found from  $\mathbf{s}_2$  or  $\mathbf{s}_1 + \mathbf{s}_2$ , if the next pair is not deleted.

Missing $\mathbf{c}_2$		Missing $\mathbf{c}_1$	
$\mathbf{c}_1$	1	$\mathbf{c}_2$	1
$\mathbf{c}_3$	01	$\mathbf{c}_3$	01
$\mathbf{c}_4$	001	$\mathbf{c}_4$	001
$\mathbf{s}_1$	xxxxxxxx	$\mathbf{s}_1$	abcxxxxxx
$\mathbf{s}_2$	11100000	$\mathbf{s}_2$	defxxxxxx
1,3,4 verified. Find $\mathbf{c}_2$ from $\mathbf{c}_1$ and $\mathbf{s}_1$ .		$\mathbf{s}_1 + \mathbf{s}_2$	1110000

In the missing  $\mathbf{c}_1$  case,  $\mathbf{s}_1 + \mathbf{s}_2$  is found easily: the streams marked "xxxxx" are identical. Thus one deletion in a span of four is filled in just by looking at two syndromes. There is no need for  $\mathbf{s}_1$  and  $\mathbf{s}_2$

any more, so the diagonalization can be restarted, if desired, at  $s_3$  after the  $c_4$  reception. Parity equations including known results need to be adjusted:

$$s_3' = s_3 - s_1 - c_3 = c_5 + c_6 \text{ and } s_4' = s_4 - c_3 - c_4 = c_5 + c_7 + c_8.$$

If both of the first pair are missing, but the next six are received correctly.  $s_1 + s_3$  verifies  $c_3, c_5$  and  $c_6$ , and  $s_4$  verifies  $c_3, c_4, c_5, c_7, c_8$ . Then  $c_1$  can be computed from  $s_2$ , and after this  $c_2 = s_1 + c_1$  is found. Four syndromes need to be looked at. Then the diagonalization can be restarted at  $s_5$  after the  $c_8$  reception

If the first 3 positions are missing, they can be found by five syndromes, provided packets 4 through 10 are received correctly.  $s_1 + s_3 + s_4$  verifies positions 4,6,7,8, and  $s_5$  verifies 5,6,7,9,10. The solution for  $c_1, c_2$  and  $c_3$  comes from  $s_1, s_2$  and  $s_3$  (the first 3 rows and columns of H):

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 + c_4 \\ s_3 + c_5 + c_6 \end{bmatrix} \quad (14)$$

A 3 x 3 binary matrix must be inverted. In parallel, the diagonalization can be restarted at  $s_6$ , where the values  $c_1, c_2, c_3$  are not needed, after the  $c_{10}$  reception.

Deletions in most cases can be filled in by looking at a small number of syndromes. Once a continuous sequence of verified and filled in packets is completed, the decoding process can be done afresh, with adjustments, starting after the end of the verifications.

It is possible to partition the deletion search even before completing verification for the current partition. To illustrate, refer to the vertical and horizontal lines of the H matrix. The upper left corner is just like the H matrix of the previously illustrated (10,3) code. This code could correct deletion of the first four packets, but deletion of packets in positions 11 and 12 would prevent this. The only verifications at this point would be 5,6,7,9,10 from  $s_5$ . One strategy could be to hold the results, and start a new diagonalization after the 10<sup>th</sup> packet. It is possible to take advantage of the verification of 9 and 10 by starting the diagonalization with  $c_9$  and  $c_{10}$ , with the syndrome comparators as  $s_1' = s_7, s_2' = s_8$ , etc. Assuming positions 13-20 are correctly received,  $s_3' = s_9$  and  $s_4' = s_{10}$  would verify 13-20 and allow simple solution for  $c_{11}$  and  $c_{12}$  from  $s_1'$  and  $s_2'$ . Then, the held diagonalization could be reevaluated, adding  $c_{11}$  and  $c_{12}$ , which would add two verification sets to allow verification of  $c_8$  and subsequent computation of  $c_1, c_2, c_3$  and  $c_4$ .

The example is a short memory code. Much more powerful convolutional codes with larger constraint length could be used. Convolutional code constraint length is generally much shorter than block code length for equivalent performance. If partition sizes comparable to constraint length are used, the minimum packet size can be greatly reduced, and the number of rows and columns used in the matrix operations would also be greatly reduced, reducing decoding complexity and the chance of false verifications.

For ordered reception, the number of consecutive packets used in a partition can extend just slightly over the range of syndromes being tested - equal to the range if there is no possibility of foreign packets or duplicate receptions. Some most recent verified packets from the prior partition that overlap with the current partition could be included, as  $\mathbf{c}_9$  and  $\mathbf{c}_{10}$  in the prior example, or their values could be subtracted from the appropriate new partition syndrome values. Also, any packet that came in after  $\mathbf{c}_{10}$  but was not verified should be included.

For out-of-order receptions, a partition may need to include a wider range of received packets, to cover early or late arrivals. Early arrivals can be taken from the set of old partition packets that did not match any verification set. Late arrivals could be included up to some maximum delay. Those that arrived beyond the maximum delay are treated as deleted.

### B. Quick-look-in Convolutional codes

Quick-look-in codes [17,18] with rate  $\frac{1}{2}$  are nonsystematic convolutional codes with good free distance and simple extraction of data. They also are convenient for use in the verification method. They are defined by  $g^{(1)}(D) + g^{(2)}(D) = D$ . Following is an example of a quick-look-in code with  $m=6$ ,  $d_{\text{free}} = 9$ , from Table 11.2 of [18].

Number data values starting with  $\mathbf{d}_1$ . The generator matrix and associated data are:

$$\begin{array}{rcccc}
 & 11 & 01 & 00 & 11 & 00 & 11 & 11 & & \mathbf{d}_1 \\
 & & 11 & 01 & 00 & 11 & 00 & 11 & 11 & \mathbf{d}_2 \\
 G = & & & 11 & 01 & 00 & 11 & 00 & 11 & 11 & \mathbf{d}_3 \\
 & & & & 11 & 01 & 00 & 11 & 00 & 11 & 11 & \mathbf{d}_4 \\
 & & & & & 11 & 01 & 00 & 11 & 00 & 11 & 11 & \mathbf{d}_5 \\
 & & & & & & 11 & 01 & 00 & 11 & 00 & 11 & 11 & \mathbf{d}_6 \\
 & & & & & & & 11 & 01 & 00 & 11 & 00 & 11 & 11 & \mathbf{d}_7 \\
 & & & & & & & & 11 & 01 & 00 & 11 & 00 & 11 & 11 & \mathbf{d}_8 \\
 & & & & & & & & & \dots & & & & & & \\
 & & & & & & & & & & \dots & & & & & 
 \end{array} \tag{15}$$

From the G matrix we know

$$\mathbf{c}_{2j+1}' + \mathbf{c}_{2j+2}' = \mathbf{d}_j. \quad (16)$$

Equation (16) is true for any quick-look-in code. Below is the H matrix and associated syndromes

$$\begin{array}{rcl}
 & 11 & \mathbf{s}_1 \\
 & 10 \ 11 & \mathbf{s}_2 \\
 & 00 \ 10 \ 11 & \mathbf{s}_3 \\
 & 11 \ 00 \ 10 \ 11 & \mathbf{s}_4 \\
 & 00 \ 11 \ 00 \ 10 \ 11 & \mathbf{s}_5 \\
 H = & 11 \ 00 \ 11 \ 00 \ 10 \ 11 & \mathbf{s}_6 \\
 & 11 \ 11 \ 00 \ 11 \ 00 \ 10 \ 11 & \mathbf{s}_7 \\
 & \quad 11 \ 11 \ 00 \ 11 \ 00 \ 10 | \ 11 & \mathbf{s}_8 \\
 & \quad \quad 11 \ 11 \ 00 \ 11 \ 00 | \ 10 \ 11 & \mathbf{s}_9 \\
 & \quad \quad \quad 11 \ 11 \ 00 \ 11 | \ 00 \ 10 \ 11 & \mathbf{s}_{10} \\
 & \quad \quad \quad \quad 11 \ 11 \ 00 | \ 11 \ 00 \ 10 \ 11 & \mathbf{s}_{11} \\
 & \quad \quad \quad \quad \quad 11 \ 11 | \ 00 \ 11 \ 00 \ 10 \ 11 & \mathbf{s}_{12} \\
 & \quad \quad \quad \quad \quad \quad 11 | \ 11 \ 00 \ 11 \ 00 \ 10 \ 11 & \mathbf{s}_{13} \\
 & \quad \quad \quad \quad \quad \quad \quad | \ 11 \ 11 \ 00 \ 11 \ 00 \ 10 \ 11 & \mathbf{s}_{14} \\
 & \quad \quad \quad \quad \quad \quad \quad \quad 11 \ 11 \ 00 \ 11 \ 00 \ 10 \ 11 & \mathbf{s}_{15} \\
 & \quad \quad \quad \quad \quad \quad \quad \quad \quad 11 \ 11 \ 00 \ 11 \ 00 \ 10 \ 11 & \mathbf{s}_{16} \\
 & \quad \quad \quad 14 & \quad \quad \quad 32
 \end{array} \quad (17)$$

Suppose the positions 15-32 ( $m+3$  paired consecutive positions, in general) are received error-free. These  $2(m+3)$  error-free positions allow immediate computation of  $2(m+1)$  consecutive data symbols ( $m+3$  from the known pairs, and  $m - 1$  just prior data symbols).

From the verifications of  $\mathbf{c}_{15}$  through  $\mathbf{c}_{32}$ , we know also  $\mathbf{c}_{15}'$  through  $\mathbf{c}_{32}'$ . Thus  $\mathbf{d}_7$  through  $\mathbf{d}_{15}$  are derived directly from these error-free values. Also, from the zero sum parity equation corresponding to row 13,

$$\mathbf{c}_{13}' + \mathbf{c}_{14}' = \mathbf{d}_6 = \mathbf{c}_{15}' + \mathbf{c}_{16}' + \mathbf{c}_{19}' + \mathbf{c}_{20}' + \mathbf{c}_{23}' + \mathbf{c}_{25}' + \mathbf{c}_{26}' \quad (18)$$

So now  $\mathbf{d}_6$  is known.

But then from row 12, now that  $\mathbf{c}_{13}' + \mathbf{c}_{14}' = \mathbf{d}_6$  is known, we can solve for

$$\mathbf{c}_{11}' + \mathbf{c}_{12}' = \mathbf{d}_5.$$

Because the packet positions in the unknown region always appear in pairs back through row 10, it is also possible to compute  $\mathbf{d}_4$ ,  $\mathbf{d}_3$ ,  $\mathbf{d}_2$  in the same way.

We can't compute  $\mathbf{d}_1$  from row 8 directly, because we don't know  $\mathbf{c}_{13}'$  and  $\mathbf{c}_{14}'$  individually.

$$\begin{array}{r}
 \mathbf{d}_1 \ \mathbf{d}_2 \ \mathbf{d}_3 \ \mathbf{d}_4 \ \mathbf{d}_5 \ \mathbf{d}_6 \ \mathbf{d}_7 \\
 \text{row 8} \quad 11 \ 11 \ 00 \ 11 \ 00 \ 10 | \ 11
 \end{array}$$

However, if just one of the pair  $\mathbf{c}_{13}$ ,  $\mathbf{c}_{14}$  gets verified, it is possible then to compute  $\mathbf{d}_1$ .

An erasure channel can correct if erasures don't cover any code word. A deletion channel with ordering is ambiguous if deletions cover within one from a code word, but succeeds if deletions don't come within one of covering a code word. Let's try a case of a quick-look-in code where there is ambiguity. Deletions of positions 1,2,4,7,8,11,12,13 come one from a code word whose 1's positions are 1,2,4,7,8,11,12,13,14. Assume there are no deletions in positions 15-32 as above. Verification sets would verify 3, 5, 6, 9, 10, but leave the nine positions 1,2,4,7,8,11,12,13,14 unverified, even though has been received correctly. In the ordered case,  $\mathbf{a}$  could be one of four possibilities:  $\mathbf{c}_{11}$ ,  $\mathbf{c}_{12}$ ,  $\mathbf{c}_{13}$ , or  $\mathbf{c}_{14}$ .

In general, there are 9 unknown packets and eight independent equations, with one of the 9 unknown packets received correctly but in unknown position. Solution for the set of alternatives might be complex, but in this quick-look-in example there is only one unknown of interest:  $\mathbf{d}_1$ . If the correct answer is  $\mathbf{c}_{13}$  or  $\mathbf{c}_{14}$ , row 8 can be used to find two different values of  $\mathbf{d}_1$ . If the correct answer is  $\mathbf{c}_{11}$  or  $\mathbf{c}_{12}$ , row 7 can be used to find two different values of  $\mathbf{d}_1$ . One of these four is correct if the packet is correct.

Assume  $\mathbf{c}_{14} = \mathbf{a}$ . Then  $\mathbf{c}_{14}' = \mathbf{a} + \mathbf{t}_{14}$ .

$$\mathbf{c}_{13}' + \mathbf{a} + \mathbf{t}_{14} = \mathbf{d}_6. \quad \mathbf{c}_{13}' = \mathbf{a} + \mathbf{t}_{14} + \mathbf{d}_6.$$

Then use row 8 to find  $\mathbf{d}_1$ .

$$\mathbf{d}_1 + \mathbf{d}_2 + \mathbf{d}_4 + \mathbf{c}_{13}' + \mathbf{d}_7 = \mathbf{0}$$

Similarly, three other values of  $\mathbf{d}_1$  can be calculated. The correct choice can be resolved by error detection built into the data. But the detection bits per data packet can be very small, possibly only one bit per packet, since the error detection can be spread over many packets, and every check bit can check about half of all the data in all the packets. Unless the channel were extremely bad, cases of covering all but one position of a code word would be rare, so it is very unlikely there would be more than one ambiguous event requiring error detection.

C. A possible partitioning

Where deletions are infrequent, verifications come with just a few syndrome tests in most cases, and the window of test frames moves quickly. Figure 9 shows a possible partitioning.

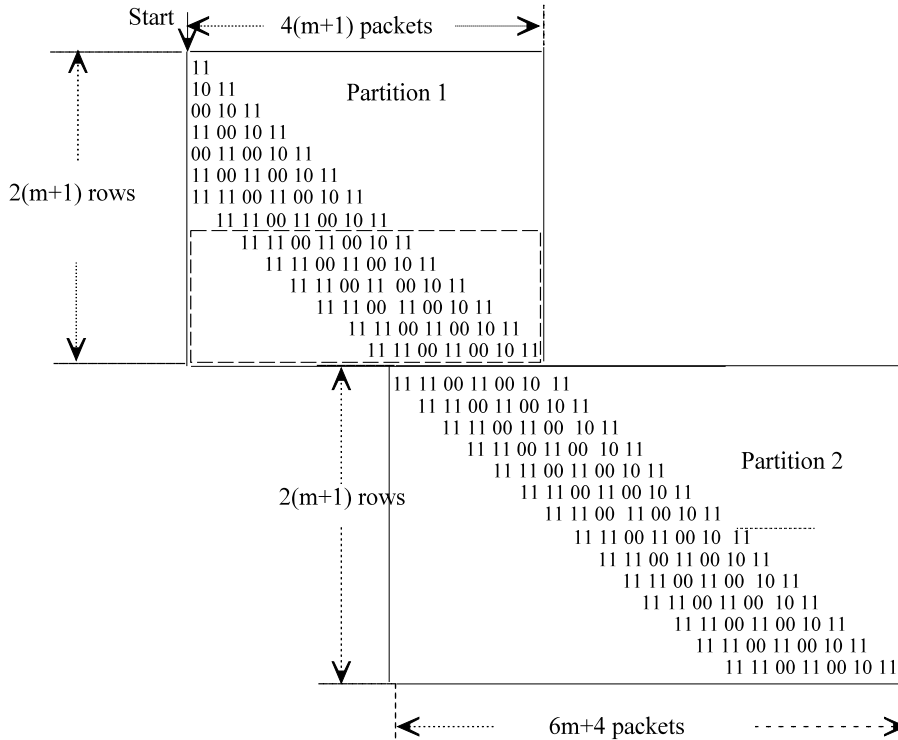


Figure 9. A possible partition for the decoder.

Although the entries are for the  $m=6$ , quick-look-in code, the same structure applies to any  $(2,1,m)$  convolutional code. The longest set of consecutive syndromes might be set at  $2(m+1)$ . The partition of  $2(m+1)$  syndromes at a start point has only about  $4(m+1)$  packets (The exact number of successive packets is uncertain due to deletions). If this fails, results are held, and partition 2 covers the next  $2(m+1)$  syndromes, corresponding to about  $6m+4$  packets. The dashed box in partition 1 indicates a range of  $m$  rows which have packets in both partitions. Verification information in this range can be passed between partitions.

Partitioning loses the ability to test against all of the  $2^{n-k}$  members of the row space of  $H$ . In fact, if  $r < n-k$  such a test would be impossible. However this loss should have little effect on

decoding power. For example, the sum of two disjoint row space members (no common positions) will be a verification set iff both members of the sum are error-free, and thus separately verifiable. If the separate verifications are made, there is no need to find the sum verification.

#### *D. Verification with long term interleaving.*

Occasionally a long terminated convolutional code or block code may fail. A simple remedy that needs small additional redundancy could be to add, for every block of  $w$  code words of  $n$  packets each, a block of  $n$  packets that is the vector XOR of the  $w$  code words. Relating to the notation in (1) and (2),

$$c_i(w+1) = \sum_{j=1}^w [c_i'(j)] + t_i, \quad i = 1, 2, \dots, n \quad (19)$$

Then, any packet position that is unverified in just one of the  $w+1$  blocks can be computed. Even more than one unsolved code word could be solved if they were unverified in disjoint places. Also, determination of one previously unverified packet of an unsolved code word could allow the code word to be solved. For example, a failure due to being one from a binary code word might be made a success if one of the verifications was part of the binary code word cover. Then the resulting new verifications could allow a new packet computation in a remote code word. A tornado code-like effect is realized.

#### *E. Use of low density codes*

Low density codes include very few packets in each syndrome. This provides early verification of some packet sets, but not those where the packets are widely spread in time. Convolutional codes are in a sense a special type of low density code, since a given check sequence generally is spread over a very short period, assuming the time to termination is far greater than the constraint length. There is, however, some merit in spreading some verification sets over a long time. This provides an interleaving effect, as just mentioned in VII.D, to help guard against clustered deletions.

An advantage in providing very few packets in each syndrome is that the chance of a false dependency is very small. For example, if each dependency includes exactly 3 packets, there are only  ${}_n C_3$  dependency combinations that could create a false verification. For large  $n$ , this is a much smaller number of possibilities as when a check equation contains about half of the packets.

The method described here is more efficient than the previously described method for finding the verification sets, and it applies to general parity check codes, including low density codes. However, low density codes are more amenable to simplification of step 4: solving for the deleted values. Solution for general parity check codes may require inverting a binary symbol matrix of size approximately equal to the number of deletions, though solution by substitution or separate deletion sets is sometimes possible. Low density codes can use the faster “tornado” effect.

## VIII. EFFECT OF PACKET BIT ERRORS

If a single packet is in error, it is extremely unlikely to be linearly dependent with the other received packets. Although it will be included in the diagonalization, it is highly unlikely to satisfy any verification set computation. In multiple path transmission, it is possible the packet will be received twice, once correctly and once incorrectly. In this case, the correct version can be observed in verification sets, while the incorrect version almost surely will not have any effect other than to have another component to diagonalize.

There is a problem, however, if two received packets have exactly the same error  $\mathbf{x}$ . Although the packets remain independent due to different random  $\{\mathbf{t}_i\}$ , some verification set that includes both can falsely verify the packets, since  $\mathbf{x}+\mathbf{x}=\mathbf{0}$ . Other sets that cover one each would not show a verification. In the ordered case where a code word is two from covered by deletions, there is a pair of packets that always appears together.

In [7], [8] and [12] the  $q$ -ary symmetric channel (qsc) is assumed, where  $q$  would be  $2^r$  in this paper’s notation. This makes errors random patterns when they occur, and the probability of cancelling errors is insignificant. However, this is not often a realistic channel model. In [7] and in [9-11] it has been suggested to do different pseudo-random transforms of different individual packets as part of the encoding, so as to make the channel look like a qsc. But this doesn’t work with unknown deletions and mis-ordering because the receiver doesn’t know which inverse transform to apply to which packet.

If the channel loses packets frequently but rarely has bit errors, the cases of cancelling bit error patterns may be rare. It may be advisable to have some crude error detection in a packet, at least to detect single and double errors. Packets with detected errors would be discarded. This would greatly reduce the chance of identical errors. Another alternative is to have no error detection in a packet, but have error detection built into the data part of the code word, as final verification that all operations were done correctly. This built-in error detection might also allow resolution of some cases that could not otherwise

be decoded uniquely. The out-of-order case where all but two positions of a code word are covered could be resolved by testing only two alternatives. This would make it as effective as the ordered case. Also, in the ordered case where the deletions cover all but one position of a code word could be solved with just a few error detection tests. This would make it as effective as the erasure-filling case where all the deletion positions are known. The overhead for inner error detection can be less than one bit per packet, since each error detection bit can depend on a pseudo-random subset of all the bits in all data packets.

## IX. CONCLUSIONS

The idea in this paper was conceived as a way to reduce computational complexity in finding verification sets with Mitzenmacher's low density parity check code method for solving deletions of packets without using sequence numbers. By using a matrix representation rather than a graph, it was found that, in addition to being an easier way of finding the verification sets, the method applies to any parity check code, low density or not. Also, instead of  $n-k$  verification sets, a whole space of  $2^{n-k}$  verification sets may be available for discovery. Thus  $H$  doesn't have to be low density because many of its row space members will be low density. The requirement that  $r > 2n-k$  by a comfortable margin is a disadvantage for large  $n$ , but it can be circumvented by partitioning verification searches. Although partitioning loses the ability to discover some verification possibilities, it appears able to discover the most important ones.

The packet-based method described allows performance with deletions, errors and mis-ordering, without sequence numbers and limited error detection, that is almost as efficient (asymptotically as efficient) as if the packets were ordered and the positions of the erroneous or deleted packets are known. Terminated convolutional codes are well-suited to the partitioning approach. However, if the packet order is completely arbitrary these simplifications might not help. In practice, out-of-order reception is not arbitrary, but is due to varying delay. It may be reasonable to treat an exceptionally delayed packet simply as a deleted packet. If it turns up in a remote partition position, it will in almost all cases look simply as a foreign packet that does not affect decoding.

The method may be important in systems that send data over multiple paths or have multiple receivers. The resultant loss of ordering can be tolerated without sequence numbering. Also, it is easily able to handle where two or more copies of a packet are received by multiple paths or from separate antennas, and only one of the copies is correct.

Use of no sequence numbering and no error detection in a packet can be a substantial overhead

saving. A compromise of using limited error detection in a packet alleviates the problem of errors adding to zero in a verification set. Alternatively, placing error detection in the data can be used to resolve ambiguity, making it closer in performance to the ideal erasure channel. A compromise of adding one or a few bits of sequence numbering can be helpful. The sequence number could be changed about once every  $x$  packets, where  $x$  is as large as a partition range- this would eliminate out-of-range receptions for consideration in a partition.

## REFERENCES

- [1] J. J. Metzner, "An improved broadcast retransmission protocol," *IEEE Trans. Commun.*, vol. COM-32, pp. 679-683, June 1984.
- [2] J. Nonnenmacher, E. Biersack, and D. Towsley, "Parity-based loss recovery for reliable multicast transmission", *IEEE/ACM Trans. on Networking*, v. 6 ,no. 4, pp. 349-361, August 1998.
- [3] B. Whetten and G. Taskale, "An overview of reliable multicast transport protocol II", *IEEE Network Magazine*, pp. 37-47, Jan/Feb 2000.
- [4] J. W. Byers, M. Luby and M. Mitzenmacher, "A digital fountain approach to asynchronous reliable multicast," *IEEE J. Sel. Areas Comm.*, v. 20, pp. 1528-1540, October 2002.
- [5] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inform. Theory*, v. 47, pp.569-584, February 2001.
- [6] J. J. Metzner, "Convolutionally encoded memory protection," *IEEE Trans. Computers*, vol. C-31, pp. 547-551, June 1982.
- [7] M. G. Luby and M. Mitzenmacher, "Verification-based decoding for packet-based low-density parity check codes," *IEEE Trans. Inform. Theory*, vol. 51, pp. 120-127, January 2005.
- [8] M.A. Shokrollahi and W. Wang, "Low-density parity-check codes with rates very close to the capacity of the  $q$ -ary symmetric channel for large  $q$ ," *Proc. IEEE International Symposium on Inf. Thy*, p. 273, 2004.
- [9] J. J. Metzner, "Majority-logic-like vector symbol decoding with alternative symbol lists," *IEEE Trans. Commun.* v. 48, pp. 2005-2013, December 2000.
- [10] J.J. Metzner and E.J. Kapturowski, "A general decoding technique applicable to replicated file disagreement location and concatenated code decoding", *IEEE Trans. Inform. Theory*, Vol. 36, pp. 911-917, July 1990.
- [11] J. J. Metzner, "Vector symbol decoding with list inner symbol decisions and dependent errors," *IEEE Trans. Commun.*, vol. 51, pp. 371-380, March 2003.
- [12] M. Mitzenmacher, "Verification codes for deletions ," *Proc. IEEE International Symposium on Inf. Thy*, p. 217, 2003.
- [13] M. Mitzenmacher, "Polynomial time low-density parity-check codes with rates very close to the capacity of the  $q$ -ary random deletion channel for large  $q$ ," *to appear in IEEE Trans. Inform. Theory*.
- [14] M. V. Davey and D. J.C. McKay, "Reliable communication over channels with insertions, deletions, and substitutions," *IEEE Trans. Inform. Theory*, Vol. 47, pp. 687-698, February 2001.
- [15] L. J. Schulman and D. Zuckerman, "Asymptotically good codes correcting insertions, deletions and transpositions," *IEEE Trans. Inform. Theory*, Vol. 45, pp. 2552-2557, November 1999
- [16] J. Justesen, "A class of constructive asymptotically good algebraic codes," *IEEE Trans. Inform. Theory*, Vol. IT-18, pp. 652-656, September 1972.
- [17] J. L. Massey and D.J. Costello, Jr., "Nonsystematic convolutional codes for sequential decoding in space application," *IEEE Trans. Commun. Tech.*, vol. COM-19, pp. 806-813, October 1971.
- [18] S. Lin and D.J. Costello, Jr., *Error Control Coding*, Prentice Hall, 1983.