ESC-TR-97-052

Technical Report
1039

A Forward Error Control Scheme for GBS and BADD

B.E. Schein
S.L. Bernstein

22 July 1997

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Lexington, Massachusetts

Prepared for the Defense Advanced Research Projects Agency
under Air Force Contract F19628-95-C-0002.

This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Defense Advanced Research Projects Agency under Air Force Contract F19628-95-C-0002.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

A FORWARD ERROR CONTROL SCHEME FOR GBS AND BADD

B.E. SCHEIN
S.L. BERNSTEIN
Group 66

TECHNICAL REPORT 1039

22 JULY 1997

LEXINGTON MASSACHUSETTS

ABSTRACT

This document provides a description of an error control scheme which can enhance the reliability of file and message transfer in the Battlefield Awareness and Data Dissemination (BADD) and Global Broadcast Service (GBS) programs.

The proposed scheme is intended to be general enough for adaptation to real network considerations, traffic profile, channel behavior, and computational and memory limitations. As such, the proposed scheme will need to be modified slightly for actual use, and some of the necessary modifications will be addressed.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. OVERVIEW

This document provides a description of an error control scheme which can enhance the reliability of file and message transfer in the Battlefield Awareness and Data Dissemination (BADD) and Global Broadcast Service (GBS) programs.

The proposed scheme is intended to be general enough for adaptation to real network considerations, traffic profile, channel behavior, and computational and memory limitations. As such, the proposed scheme will need to be modified slightly for actual use, and some of the necessary modifications will be addressed.

1.1 Channel and Network Description

BADD (Battlefield Awareness and Data Dissemination) is a DARPA ACTD (Advanced Concept Technology Demonstration) program which is developing information management techniques for the selection, integration, and addressing of data needed by the warfighter. BADD will make use of the GBS (Global Broadcast Service) satellite system which will be able to convey high data rate streams (approximately 24 Mbps) to small terminals. Low rate reverse links using the "Tactical Internet" will permit some users to request or "pull" selected information and provide a means for two-way protocol closure, e.g., ARQ for reliable message reception. However, it can not be assumed that all users will have access to a return link. Hence strong and efficient forward error control is essential. This Report will present a particular forward error control scheme designed for message recovery by end users.

At the physical layer, GBS will employ highly efficient modulation and coding techniques yielding a very low bit error rate (on the order of 10^{-10}) at low signal-to-noise ratio per bit. However, even at these low error rates there are still several potential situations that can interfere with reliable message transmission. Some examples of these situations are as follows: First, long messages (e.g., images consisting of several gigabytes) take minutes to transmit and would have a non-negligible probability of containing at least one random error. Second, the line-of-sight from the GBS satellite to a user can be disturbed by discrete events such as a soldier walking in front of the antenna. Finally, even with an undisturbed RF channel, a steady stream of high rate data being delivered to a busy end-user computer through numerous hardware and software layers can result in dropped cells or packets.

In order to mitigate these problems, a powerful forward error correcting and erasure filling scheme was designed for use with BADD and GBS. The goal was to recover dropped data packets and correct occasional packet errors while using minimal computation and coding overhead. Since a single user may not receive a steady stream of data, the scheme was designed to be implemented on a per-message basis. For practical reasons, the scheme was designed to be implemented at the application level in software, without the need for additional hardware. Because of this, the minimization of computational complexity was a primary design constraint.

Since this scheme is implemented on a per-message basis, and since the general error event may result in the loss of a large amount of data (possibly on the order of a few megabytes), the scheme is designed for the protection of large files. With the type of error events considered, no per-message scheme can protect small messages in the absence of a reverse link when low latency and low coding overhead is desired.

The scheme is based on multidimensional Reed-Solomon product codes with a sub-optimal but computationally feasible decoding procedure. The encoding and decoding procedure will be described in detail. A summary analysis of its capabilities and a discussion of the sub-optimality of the decoding procedure will be presented. A prototype encoding and decoding procedure was implemented in C, and the results of prototype benchmark testing will be presented.

1.2 Current Error Control Scheme

The current error control scheme used with BADD and GBS is simple but inefficient. The scheme uses a repetition code, where the number of times a message is repeated is variable. A message is broken into suitably sized packets, and the entire message is repeated. For decoding, the first packet which passes its checksum is accepted and all others are discarded.

This error control scheme has several advantages. First, it is simple to analyze and implement. Second, it requires minimal computation. It uses only a checksum in addition to the standard network checksums of UDP, IP, and the convergence sub-layer of ATM. Third, it requires virtually no additional memory since only original data is stored, along with a record of which packets have been received. Fourth, the probability of failing to receive a message can be made arbitrarily small using a sufficient number of repetitions. Finally, the application can ignore future message repetitions once it has accepted a complete set of packets. Thus, if the channel is good for a long time, such that the entire message is successfully transmitted the first time, there is virtually no delay before the message is successfully received and decoded.

This error control scheme has several disadvantages, however. First, and most importantly, it is inefficient. The error control imposes 100% overhead for each repetition (this is in addition to standard per-packet network overhead). Second, the scheme uses sub-optimal decoding in accepting the first packet which passes its checksum. Thus, although unlikely, error-filled packets may pass the checksum and not be rechecked. From a data-integrity point of view, a simple improvement would be to use a majority-rule system to virtually eliminate the chance of accepting a packet with errors (which is only an issue if it is the first packet to pass its checksum). This "improvement," however, comes with a severe delay penalty. Third, a single dropped packet causes a packet decoding latency at least equal to the transmission time of the entire message. This can be unacceptable, particularly when dealing with long messages (consider dropping one of the last packets of a message). Fourth, channel outages (erasure bursts) may drop all copies of short messages unless artificial delay or padding is introduced.

1.3 Design Goals for Proposed Scheme

The error control scheme is designed to meet the following goals:

1. Small coding overhead — 10 to 15% (plus network overhead).
2. Minimal decoding computation.
3. Reasonable memory requirement.
4. Recover from long channel outages as well as scattered packet losses and occasional errors.
5. Low latency when the channel is behaving well.
6. Decode simultaneously with data arrival to reduce decoding delay.

This scheme is designed to recover missing data due to lost packets, as opposed to recovering data from noisy signals on a typical physical channel. Packets with byte errors are rarely delivered to an application, since multiple network checksums (from UDP, IP, and the convergence sub-layer of ATM in the envisioned network) are used. The two major advantages of dealing with packet erasures rather than actual errors is that the locations of data erasures are precisely known and, in the absence of errors, the decoding procedure can determine apriori what corrective steps will recover the lost data, if at all possible.

2. DESCRIPTION OF THE CODING SCHEME

2.1 General Code Description

In general, a UDP packet will either be received correctly or be entirely erased due to checksum failure somewhere within the network protocol stack (ATM, IP, or UDP). The network does not notify the decoder (application) when a packet fails its checksum.

The scheme we propose therefore imposes no correlation (beyond the network checksums) amongst the bytes of the individual UDP packets. We use a set of identical codes involving a series of UDP packets. We use a codeword for each data byte position in the UDP packets. This set of "parallel" codewords is intended to fill in the packet erasures due to checksum failure and additionally to recover from occasional (very rare) packet errors.

The dimensions and packets are arranged qualitatively as pictured in Figure 1. Each cell corresponds to a single code symbol, and each code symbol is one byte. The bytes of a UDP packet are enumerated vertically from top to bottom, along Dimension 0. Code 1 spans Dimension 1, and similarly Codes 2 and 3 span Dimensions 2 and 3, respectively. Each packet can be identified with a triplet (D1,D2,D3) corresponding to its logical position in the figure.

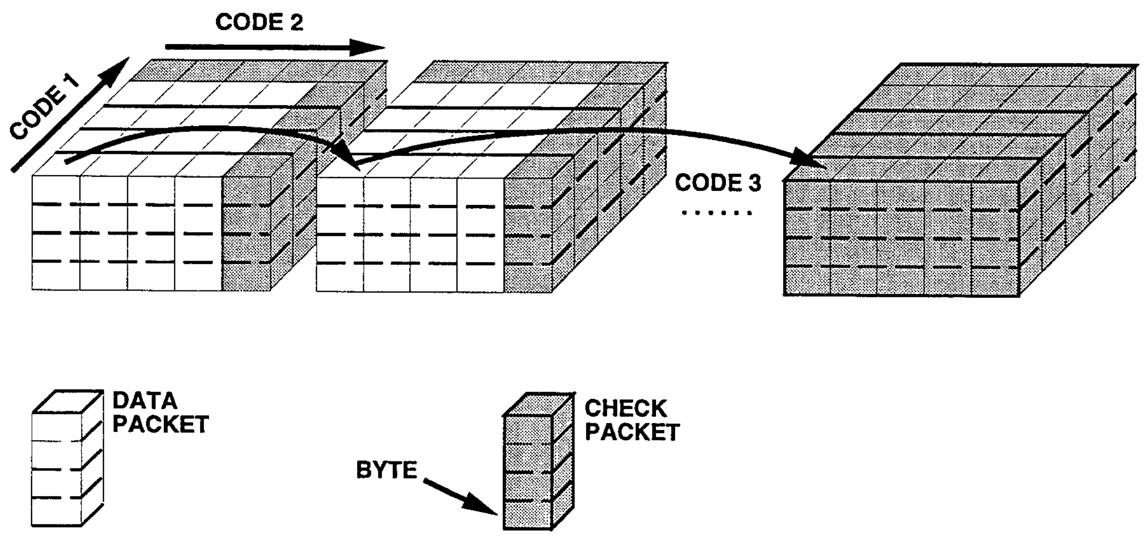

Figure 1. Three-dimensional product code structure.

To see how the parallel set of codewords are arranged, refer to Figure 2. The bytes of UDP packets are enumerated along Dimension 0 (d0). Each codeword in the set fills in the erasures in a single byte position along Dimension 0. Ignoring packet errors for the moment, a specific pattern of packet erasures will result either in successful erasure recovery by every codeword in the parallel set or failure by every codeword in the parallel set.

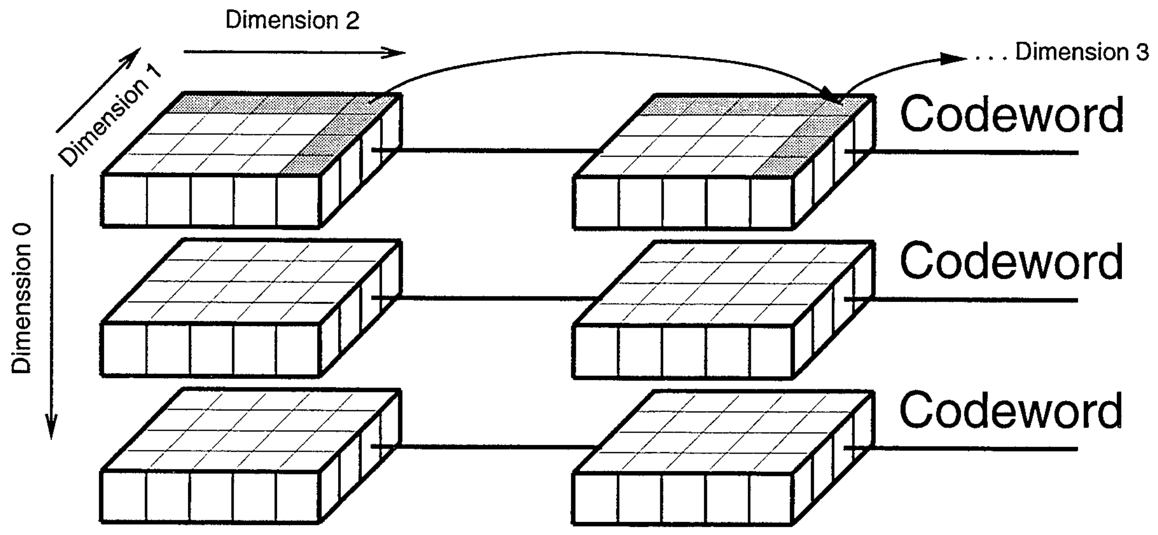

Figure 2. Parallel codeword structure.

The specific code we propose for erasure recovery (each Codeword in Figure 2) is a 3-dimensional product code, using shortened Reed-Solomon codes as the constituent codes. Using a straightforward, sub-optimal decoding procedure for the product codes to minimize computation, the 3-dimensional product code yields good burst erasure and random erasure recovery as well as reasonable random error and erasure recovery. The decoder performance will be addressed in Section 3.

Recall that an (n, k) block code maps k information symbols into n codeword symbols. A Reed-Solomon code (RS code) is a block code with minimum distance $\mathrm{d_{min}} = \mathrm{n} - \mathrm{k} + 1$. Such codes are called maximum-distance codes; no block code with the given (n, k) can have a larger minimum distance. An optimal decoder (minimum distance decoder) for any block code with minimum distance $\mathrm{d_{min}}$ can guarantee correction of any pattern of ν erasures and t errors provided $\nu + 2 * t < \mathrm{d_{min}}$, and can additionally correct some patterns of errors and erasures that violate this inequality. Using reasonable assumptions about the physical network, a minimum distance decoder is a maximum-likelihood decoder. Alternately, a decoder that decodes up to the minimum distance of the code will guarantee correction of any pattern of errors and erasures that satisfies this inequality, but it is free to decode arbitrarily (or give up) if the inequality is not satisfied.

The constituent shortened RS codes we use are subspaces of RS codes corresponding to setting a number of the k information symbols, say the first l of them, equal to zero. We use l fewer symbols per codeword. The shortened RS code is then an $(n - l, k - l)$ maximum-distance block code. The minimum distance of a shortened RS code is the same as that of the parent RS code. Note that using a shortened RS code decreases the information rate of the block code since $\frac{k-l}{n-l} < \frac{k}{n}$, and thus increases the coding overhead relative to the full-length code.

A decoder guaranteeing correct decoding up to the minimum distance of a shortened RS code (addressing both errors and erasures) can be implemented entirely with lookup tables and some basic arithmetic operations, thereby minimizing computation. However, the size of the lookup tables is practical only for extremely weak RS codes — where the minimum distance is small. We address this in Section 2.3.1.

For efficient implementation, reasonable arithmetic and syndrome table sizes, and high code rate, we use RS codes defined over the 256-member Galois Field, denoted GF(2^8) or GF(256). The most important factor in deciding the field size is efficient implementation since the error control scheme must be implemented at the application level — without dedicated hardware for efficient implementation of symbols with size unequal to 8 bits (or 16). All our shortened RS codes are based on a single, full-length $(255, 253)$ RS code, so the codeword and information symbols are 8 bits each. The minimum distance of this and its shortened codes is 3. We encode data bytes systematically (losing no performance), and thus a shortened RS codeword of length LEN consists of (LEN-2) information symbols, $\{i_k\}_{k=0}^{LEN-3}$, and two parity-check symbols $pc_{0,1}$. The qualitative location of the information symbols and the parity check symbols is diagrammed in Figure 1, where only a few information symbols and a single parity check symbol (shaded) per dimension is pictured for compactness. The computationally simple (shortened) RS decoder used can guarantee correct decoding of any single error or up to two erasures, which is the most that an optimal minimum-distance decoder can guarantee.

We build a product code out of weak constituent RS codes to provide a more powerful overall (block) code with a decoder still relying largely on lookup tables. We use a 3-dimensional product code with constituent codewords oriented along Dimensions 1, 2, and 3, as qualitatively diagrammed in Figure 1. The defining quality of a product code is that the symbols oriented along a specific dimension (1, 2, or 3) form a codeword in that dimension's block code — these dimensional block codes are called the constituent codes of the product code. We allow the lengths of the shortened RS codes to vary according to message length or application requirements (limitted memory, delay, etc.) — all the codewords (rows) oriented along D1 must be the same length, but this length may be different than those oriented along D2 or D3.

The minimum distance of an n-dimensional product code is the product of the minimum distances of its constituent codes [1]. The 3-d product code we use has $d_{\min} = 3^3 = 27$. We use a sub-optimal decoding algorithm (not a minimum-distance decoder for the product code) which, in the absence of errors, still decodes the 3-d product code up to its minimum distance. The sub-optimal decoder is discussed in Section 2.3.

2.2 Encoding Procedure for the 3-D Product Code

For concreteness, assume that the lengths of the constituent RS codes along Dimensions 1, 2, and 3 equal LEN1, LEN2, and LEN3, respectively. We use shortened RS codes based on a systematically-encoded (255,253) code, so every constituent codeword contains 2 parity-check symbols. The packets representing the last two positions along Dimension 3 (D3), for example, consist entirely of parity-check symbols for the parallel 3-d product codewords. In fact, every packet with sequence number (i.e., position along the appropriate dimension) D1sn $>$ (LEN1 $-$ 3), D2sn $>$ (LEN2 $-$ 3), or D3sn $>$ (LEN3 $-$ 3) consists entirely of parity-check symbols for codewords oriented along D1, D2, or D3, respectively (enumeration in all dimensions begins with 0).

Refer now to Figure 3, where d0 is oriented directly into the page (a bird's-eye view of Figure 1). In the figure, the code dimensions are LEN1 = 5, LEN2 = 6, and LEN3 = 5. The packets consisting of parity-check symbols are shaded in the figure. The origin is located at the lower left corner. The packets are numbered according to the order in which they are sent across the channel (no interleaving is used within the encoded message).

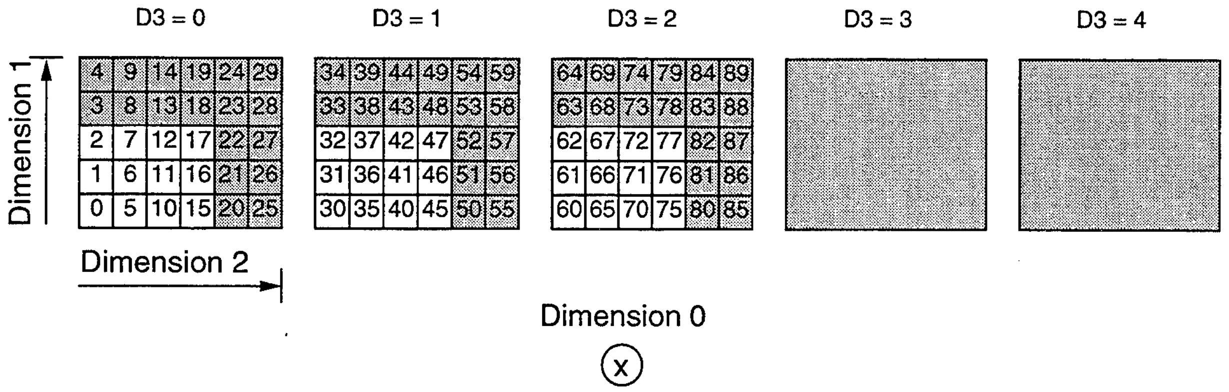

Figure 3. Product code packet ordering.

Encoding can be accomplished by logically arranging the original data accordingly and systematically encoding the constituent codes in each dimension.

2.3 Decoding Procedure for the 3-D Product Code

The sub-optimal, non-adaptive decoding procedure we suggest is straight-forward. First decode the shortened RS codewords oriented along Dimension 1. Then decode those oriented along Dimension 2. Finally, decode those oriented along Dimension 3. We do this in parallel along Dimension 0, the bytes of the UDP packets.

In the absence of errors, this sub-optimal decoding procedure decodes up to the minimum distance of the product code. We characterize the sub-optimality below. We do not envision this full sub-optimal decoding procedure to be run on every 4-d data set, since it is unnecessary (and computationally wasteful) when the channel is behaving well. The most interesting and useful quality of this error control scheme is its adaptability to channel conditions (packet erasures). The advantage of primarily filling erasures rather than correcting errors lies in knowing a priori exactly what decoding steps must be taken to fully recover the data. A simple but effective approach could be to decode a subset of the 3 dimensions. Depending upon the erasure pattern, it is likely that at most two or possibly one of the three constituent codes need be decoded – this would reduce the core decoding computation by $\frac{1}{3}$ or $\frac{2}{3}$ if such an adaptive scheme were used. The computational savings over the full decoding procedure will be determined precisely by how aggressive an adaptive approach is used and by how conservative an approach is desired when considering the presence of additional, though rare, packet errors.

2.3.1 Single RS Codeword Correction Technique

Consider the full decoding procedure described above. An efficient computational algorithm for implementing the procedure, such as the one we chose to implement for preliminary benchmark testing, may opt to compute all the appropriate syndromes for every shortened RS codeword in every dimension as the data arrives, filling in all erasures with zeros. This allows a single pass over the 4-d data set rather than requiring a pass over the 4-d data set for every dimension in the product code. If a one-pass algorithm is used (as we suggest), corrections (both erasures and errors) to constituent codewords contribute to the syndromes for constituent codewords that will be decoded later in the procedure. Thus an efficient method (time and memory) for updating syndromes corresponding to constituent codewords that will be decoded later is crucial.

Single RS codeword syndromes are computed by multiplying the received word by the parity-check matrix, filling in all erasures with zeros for the computation. The adjustment of syndromes in all remaining undecoded dimensions is easy — when we fill in an erasure, we need only multiply the erasure value by the proper row of the parity check matrix and add the result to the cumulative syndrome computations. Error corrections are handled similarly by multiplying the error value (correct symbol minus the error symbol) by the proper row of the parity check matrix and adding. The core syndrome computation using this method requires 2 multiplications and 2 additions per data byte per dimension.

We generate a syndrome lookup table with a number of entries equal to $(2^8)^2$ holding $(2^8-1)^2$ valid entries (the remaining entries are 0). The table is generated by multiplying every possible single error word against the parity-check matrix to yield the syndromes — there are 255 possible non-zero error values in 255 possible positions. The two syndrome symbols become the keys of the table, while the entry stores the error value and location. Since the minimum distance of the RS code is greater than 2, we are guaranteed to generate a unique pair of symbols (the syndrome) for every single-error word.

Error and erasure correction is implemented differently depending on the number of erasures in the codeword. When we have 0 or 1 erasure, we use the syndrome lookup table mentioned above.

Consider a codeword without erasures. We are guaranteed by the minimum distance of the RS code (equal to 3) to be able to correct any single error in the codeword. If there is a single error, the syndrome symbols will not both be zero. As mentioned above, the syndrome table then necessarily stores the correct error value. Now if the codeword has multiple errors, we have no bahavioral guarantee — either the syndrome will correspond to an invalid entry, or it will correspond to a codeword with distance at least 3 from the original.

As an aside and for completeness, note that the shortened codes have a small advantage over the full-length codes in the presence of multiple errors. For the shortened codes, erroneously determining that the syndrome corresponds to a "detected" single error may result in a "detected error location" outside the bounds of the shortened codeword. For example, consider a received vector (shortened codeword plus noise) of length LEN ¡ 255 symbols with two errors and no erasures. Considering how the syndrome table is generated, the computed syndrome may correspond to a single error location *in a full-length RS codeword.* Then with probability $\frac{255-\text{LEN}}{255}$, the "detected error location" is not a valid location within the shortened codeword. This is a very small advantage considering the incredibly unlikely event of receiving multiple symbol errors in a constituent codeword.

Now consider a codeword with a single erasure. Without additional errors, we are guaranteed to find the syndrome in the table because this erasure corresponds to a single error whose value is the proper (erased) codeword symbol. With a single additional error (i.e., 1 error and 1 erasure), we are guaranteed not to find the syndromes in the table and thus we will necessarily, knowingly fail to decode the codeword. This is guaranteed by the minimum distance of the code. If we found a syndrome table entry whose error location corresponded to the erased location, then filling in the erased location with the error value would result in a valid shortened codeword with distance 2 from the correct codeword. Thus, with one erasure and one error, the table entry corresponding to the syndrome will either contain an error location unequal to the known erasure location or will be an invalid entry (zero).

Finally consider a codeword with two erasures. Assume the two erasure locations are j, k, assume $j < k$, and assume there are no additional errors. Assume that the codeword $\mathbf{c} = [c_0 c_1 \cdots c_n]$ is sent and that the received vector is $\mathbf{r}$, where the (possibly shortened) code has n symbols. Denote the $n \times 2$ parity-check matrix by $\mathbf{H}^\star$, and denote the i^{th} row of $\mathbf{H}^\star$ by $\mathbf{h}_\mathrm{i} = [h_{i,0}\ h_{i,1}]$. We can rewrite $\mathbf{c}$ as

$$\begin{aligned}
\mathbf{c} &= [c_0 c_1 \cdots c_n] \\
&= [c_0 c_1 \cdots c_{j-1} 0 c_{j+1} \cdots c_{k-1} 0 c_{k+1} \cdots c_n] \\
&\quad + [00 \cdots 0 c_j 0 \cdots 0] \\
&\quad + [00 \cdots 0 c_k 0 \cdots 0] \\
&= \mathbf{c}' + \mathbf{c}_\mathrm{j} + \mathbf{c}_\mathrm{k}.
\end{aligned}$$

Now filling in the erased locations with zeros, we can similarly rewrite the received vector (without errors) as

$$\begin{aligned}\mathbf{r} &= \mathbf{c}' \\ &= \mathbf{c} + \mathbf{c}_{\bar{\mathbf{j}}} + \mathbf{c}_{\bar{\mathbf{k}}} \\ &= \mathbf{c} + \mathbf{c}_{\mathbf{j}} + \mathbf{c}_{\mathbf{k}},\end{aligned}$$

where $\mathbf{c}_{\bar{\mathbf{j}}} = [00\cdots0\,-c_j0\cdots0] = \mathbf{c}_{\mathbf{j}}$ since every element is its own additive inverse (in any GF of characteristic 2). Denoting the syndrome $\mathbf{v} = [v_0\ v_1]$, we can write

$$\begin{aligned}\mathbf{v} &= \mathbf{r}\mathbf{H}^\star \\ &= \mathbf{c}_{\mathbf{j}}\mathbf{H}^\star + \mathbf{c}_{\mathbf{k}}\mathbf{H}^\star & (1) \\ &= [c_j\ c_k]\begin{bmatrix}\mathbf{h}_{\mathbf{j}} \\ \mathbf{h}_{\mathbf{k}}\end{bmatrix} \\ &= [c_j\ c_k]\,\mathbf{H}_{\mathbf{sub}}, & (2)\end{aligned}$$

where (1) follows since $\mathbf{c}\mathbf{H}^\star = 0$ for any codeword.

Now we prove by contraposition that we can find c_j, c_k. If $\mathbf{H}_{\mathbf{sub}}$ is not invertible, it has a non-trivial null-space [1, Ch2]. Then with the syndrome $\mathbf{v}$ fixed, there are multiple pairs $[x\ y]$ satisfying (2) for $[c_j\ c_k]$. Then we cannot uniquely decode this codeword. But, by assumption, this codeword has 2 erasures, no errors, and the code has minimum distance 3 $\rightarrow\leftarrow$. Thus $\mathbf{H}_{\mathbf{sub}}$ is invertible. Furthermore, $\mathbf{H}_{\mathbf{sub}}^{-1}$ is unique [1]. We can thus always invert $\mathbf{H}_{\mathbf{sub}}$ to solve uniquely for c_j, c_k. This extends to higher dimensions for more powerful RS codes, though inverting higher-dimensional matrices requires significantly more computation (e.g., solving for 2 erasures requires 10 multiplications, 2 additions; solving for 3 erasures requires 39 multiplications, 17 additions using the Laplace expansion and cofactors for inverting the matrix — this may still be acceptable).

Finally, consider a codeword with 2 erasures and one or more additional errors. Using the method of multiplying $\mathbf{v}$ by $\mathbf{H}_{\mathbf{sub}}^{-1}$, we will never know whether there are additional errors. We will, in general, replace the erased symbols with incorrect symbols.

3. DETERMINISTIC ERASURE AND ERROR PERFORMANCE

We have characterized the deterministic performance of the scheme in a number of ways. Since we have no reasonable stochastic channel model, we have not attempted any stochastic analysis. We could use a coarse Gilbert-Elliot type channel model in a stochastic analysis, but this would be an arbitrary choice. This coarse model would represent a 3-dimensional space (a possible basis would be the two average state dwell times and the average time in one of the two states), and an analytical mapping could be approximated only in limited regions of the space. We feel that such analysis should be attempted after an approximate model is developed for the actual channel.

3.1 Sub-Optimal Decoding Loss

Considering only erasures, we can explicitly characterize erasure patterns that will be undecodable under the specific sub-optimal decoding scheme we have chosen and under the full-power decodability of the product code.

Exploiting the full power of the code, i.e., using an optimal (minimum-distance) decoder, any undecodable erasure pattern must contain a $3 \times 3 \times 3$ fragmented cube of erasures within the 3 - d product code. A fragmented cube of erasures can be defined mathematically as follows. For notational purposes, each logical position in the data set can be identified by a triplet (D1sn,D2sn,D3sn) identifying the position along dimensions D1, D2, and D3, respectively. A fragmented cube of erasures consists of exactly 27 erasures whose corresponding set of 27 triplets $\{(\text{D1sn}, \text{D2sn}, \text{D3sn})\}$ satisfies the following criterion:

- there are exactly 3 unique sequence numbers in the first coordinate, D1sn_1, D1sn_2, and D1sn_3, each appearing exactly 9 times, AND
- there are exactly 3 unique sequence numbers in the second coordinate, D2sn_1, D2sn_2, and D2sn_3, each appearing exactly 9 times, AND
- there are exactly 3 unique sequence numbers in the third coordinate, D3sn_1, D3sn_2, and D3sn_3, each appearing exactly 9 times.

This $3 \times 3 \times 3$ fragmented cube of erasures corresponds to a set of 9 erasures in a D1 $\times$ D2 plane repeated in 3 different positions along D3 (not necessarily successive positions). The pattern of 9 erasures must consist of exactly 3 rows and 3 columns (not necessarily successive rows or columns) each with 3 erasures to form a 3×3 array. Refer to Figure 4 for such a $3 \times 3 \times 3$ fragmented cube of erasures. This is a bird's-eye view of the 4-d data set with Dimension 0 directed into the page. Each large **X** represents an erasure.

As explained in Section 2.3.1, any single codeword containing fewer than 3 erasures will be correctly filled. Thus, after simply iterating the decoding procedure multiple times (running through

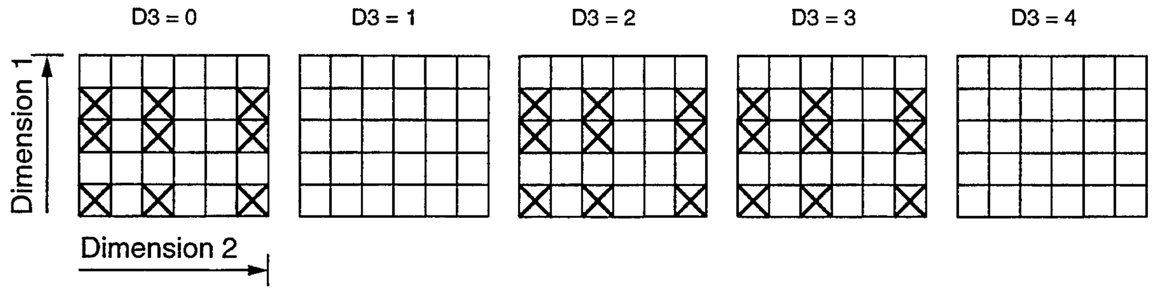

Figure 4. 3 x 3 x 3 fragmented cube of erasures.

the full decoding procedure multiple times on the same data set), only constituent codewords with more than 3 erasures could be left unfilled. Since this is true for every dimension of the code, every undecodable erasure pattern using the iterated technique must contain a $3 \times 3 \times 3$ cube of erasures. Furthermore, in the absence of errors, the iterated procedure monotonically decreases the distance between the received word and the correct codeword (it does not introduce erroneous symbols in the absence of received symbol errors), and thus any erasure pattern correctable by the iterated procedure will be correctable by a minimum-distance decoder. Conversely, any $3 \times 3 \times 3$ cube of erasures is undecodable (equidistant from 2 codewords), and thus every erasure pattern containing such a cube is undecodable for any decoder.

Now consider the sub-optimal decoding procedure described in Section 2.3. Describing the undecodable erasure patterns is difficult, but the patterns are fully characterizable. We will argue that any undecodable erasure pattern must contain three sub-patterns of 9 erasures in a $D1 \times D2$ plane, each sub-pattern with a different index in D3. Each sub-pattern of 9 erasures must contain 3 different columns with 3 erasures each, each column with an erasure in at least 1 common D2 row. Call such a row a cross-pattern row. Then, finally, each of the three sub-patterns with 9 erasures must have at least 1 commonly erased (D1,D2) pair within their cross-pattern rows (to yield a D3 codeword with 3 erasures).

Refer to Figure 5 for such an undecodable pattern when using the simple sub-optimal decoding procedure. This is a series of bird's-eye views of the 4-d data set with D0 directed into the page. Each large **X** represents an erasure. All cross-pattern rows are highlighted. The top sketch depicts an original, hypothetical erasure pattern on the 4-d data set. The middle sketch depicts the three sub-patterns of 9 erasures. The bottom sketch shows the undecodable D3 codeword with 3 erasures that must be left after the first 2 dimensions have been decoded. This is an undecodable erasure pattern.

Returning to the characterization of erasure patterns, consider any erasure pattern that satisfies these conditions. Focusing on the $D1 \times D2$ plane, only an erasure pattern containing such a 9-erasure sub-pattern can be left unfilled after decoding the first two dimensions of the product code. Indeed,

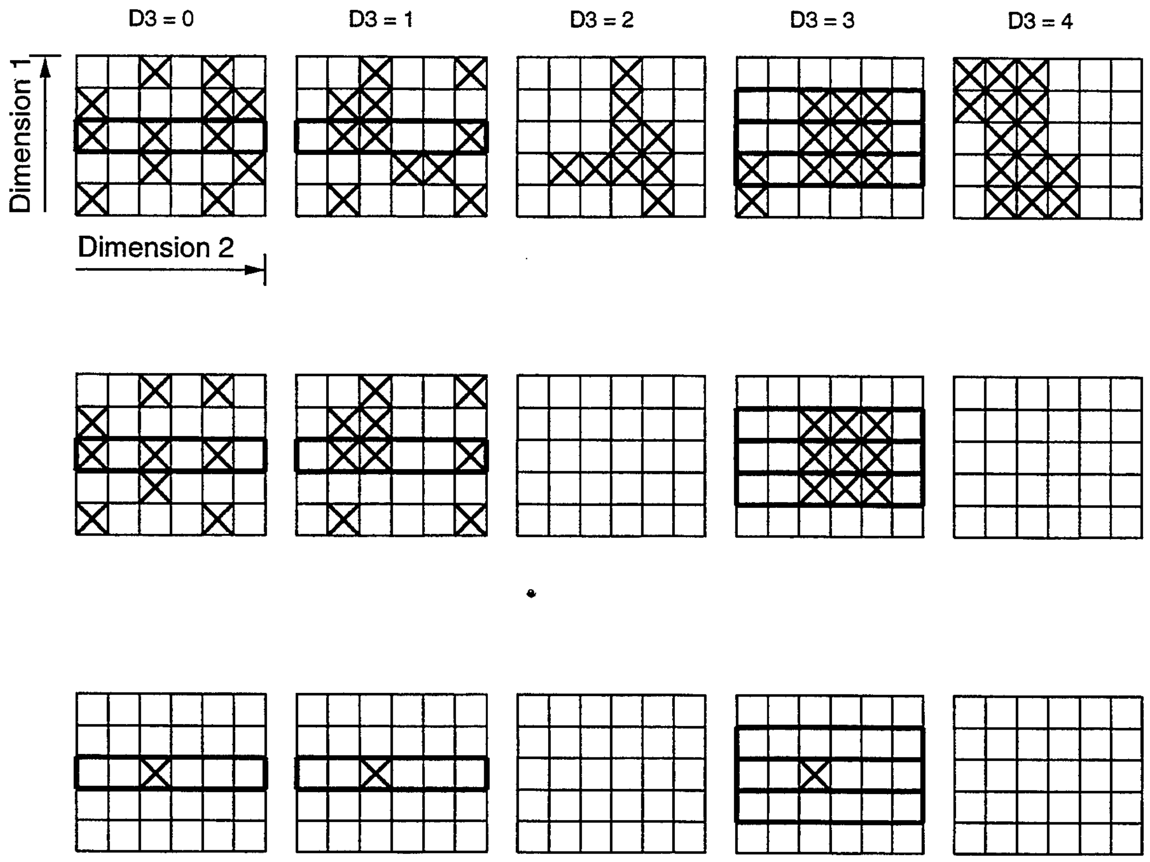

Figure 5. Undecodable erasure pattern using the sub-optimal decoding method.

the cross-pattern rows will be left unfilled after decoding the first two dimensions. Focus back on the full 4-d erasure pattern and work backwards in the decoding procedure. Only patterns with 3 or more remaining erasures along a D3 codeword will be left unfilled after decoding D3. Any triplet of erasure sub-patterns with a commonly erased (D1,D2) pair in their cross-pattern rows will leave a D3 codeword at this location with at least 3 remaining erasures, and thus every erasure pattern containing such a triplet of sub-patterns is undecodable. By a similar argument, any undecodable erasure pattern must contain such an erasure sub-pattern.

Note that this class of erasure patterns contains the $3 \times 3 \times 3$ fragmented cubes (i.e., the erasure patterns that are undecodable for a minimum-distance decoder), of course, as all 3 rows of each sub-pattern are cross-pattern rows and the 3 sub-patterns overlap in all 9 (D1sn,D2sn) pairs (leaving 9 undecodable D3 codewords).

Such characterization of undecodable erasure patterns for both an optimal and this particular sub-optimal decoding procedure will ease stochastic analysis. Further, this characterization will make stochastic performance simulation (if desired) simple for an optimal decoding procedure. Finally, it may help simplify simulation of the sub-optimal decoding procedure, though this could just as easily be obtained by generating erasure patterns and simply correcting (in the appropriate order) all codewords with fewer than 3 erasures.

3.2 Performance Guarantees

We summarize several deterministic performance characterizations. For notation, we have defined the code dimensions $n_1 = \text{LEN1}, n_2 = \text{LEN2}$. In the following tables, a single burst of erasures refers to a single burst across the entire 4-d data set. Similarly, a double burst corresponds to 2 separate bursts (possibly contiguous) across the entire 4-d data set. Furthermore, a random erasure per cube corresponds to a single random erasure for every D1 $\times$ D2 plane.

We took a game-theoretic point-of-view to generate the following performance guarantees. Specifically, we found the absolute worst orientation of burst erasures, random erasures, and random errors such that we can still guarantee perfect data recovery using the sub-optimal decoding procedure. Allowing the "Maximum Erasure Burst Size" to increase by one will admit an erasure/error pattern that will break the decoding procedure (i.e., the decoding procedure will fail to produce an error- and erasure-free 4-d data set). The code will perform significantly better under reasonable stochastic variation.

For specific numerical values, we assumed $n_0 = 400$ (i.e., 400 bytes per packet), $n_1 = n_2 = 69$, and $n_3 = 25$.

TABLE 1

Single Burst Plus Random Erasure Protection Guarantees

Max. # Random Erasures Per Cube (Dropped Packets)	**Max. Erasure Burst Size (Packets)**
0	$n_1(2n_2) = 9522$ (3.0 second outage at 10 Mbits/sec)
1	$n_1(2n_2 - 1) + 3 = 9456$
2	$n_1(2n_2 - 1) + 1 = 9454$
3	$n_1(n_2 + 3) = 4968$
4	$n_1(n_2 + 2) + 3 = 4902$
5	$n_1(n_2 + 2) + 1 = 4900$
6	$n_1(n_2 + 1) = 4830$
7	$n_1(n_2) + 3 = 4764$
8	$n_1(n_2) + 1 = 4762$
≥ 9	0

TABLE 2

Double Burst Plus Random Erasure Protection Guarantees

Max. # Random Erasures Per Cube (Dropped Packets)	Max. Erasure Burst Size (Each)
0	$n_1\,(n_2) = 4761$ (1.5 second outages at 10 Mbits/sec)
1	$n_1\,(n_2 - 1) + 2 = 4694$
2	$n_1\,(n_2 - 1) + 1 = 4693$
3	$n_1\,(2) = 138$
4	$n_1 + 4 = 73$
5	$n_1 + 2 = 71$
6	$n_1 = 69$
7	2
8	1
≥ 9	0

TABLE 3

Single Burst Plus Random Erasure Guarantees, Given a Single Random Error

Max. # Random Erasures Per Cube (Dropped Packets)	Max. Erasure Burst Size (Packets)
0	$n_1\,(n_2 - 1) + 1 = 4693$
1	$n_1 = 69$
2	3
3	1
≥ 4	0

TABLE 4

Double Burst Plus Random Erasure Guarantees, Given a Single Random Error

Max. # Random Erasures Per Cube (Dropped Packets)	Max. Erasure Burst Size (Each)
0	3
1	2
2	1
≥ 3	0

3.3 Benchmark Tests

An encoder/decoder pair was programmed in C for proof-of-concept and for preliminary benchmark testing. This code was not optimized (at the programmer's level) for either speed or storage requirements. Several timing tests were run on the implementation. The following decoding times were measured across the core decoding procedure only (after loading received data into RAM and before writing out to file). The times do not account for receiving the data and transferring to RAM. Further, we saw a significant reduction in decoding time when the decoder was run twice in succession. We attribute this to caching of the program (not the data).

The data set had the following dimensions: $397 \times 69 \times 69 \times 25$ encoded bytes, corresponding to $397 \times 67 \times 67 \times 23$ bytes of real data. A packet size of 400 bytes was chosen, where 3 bytes are used as packet sequence numbers (hence the dimension size 397 above). The second-run times are recorded here (real time), as well as the corresponding rate of information data (not codeword data) processed:

Sun Sparc 10 (unloaded):	144.0 seconds (± 1)	2.3 Mbits/sec
Sun Ultra 1 (unloaded):	27.2 seconds (± 1)	12.1 Mbits/sec

We ran the decoding tests on perfect data sets (no erasures, no errors). We do not predict a significant increase in decoding time when faced with a significant quantity of erasures; filling in erasures once the syndromes have been computed requires little additional computational effort. In the interest of time, we opted to implement the full decoding procedure described in Section 2.3 regardless of the erasure pattern.

As discussed in Section 2.3, the scheme was designed to allow fast decoding when the channel is behaving well. For example, decoding only the first dimension will generally correct low-weight, non-bursty erasure patterns. See Section 2.3 for further discussion. The data rate numbers recorded above do not reflect an intelligent (adaptive) decoder implementation.

4. IMPLEMENTATION ISSUES

4.1 A Few Words on Interleaving

At first glance (and intuitively), it appears that interleaving the data over the channel would improve the burst erasure protection for our sub-optimal decoding scheme. However, interleaving within a 4-d data set yields no gain. To focus on the issue, we restrict our attention to a single burst of erasures without any additional packet erasures or errors.

We decode the data set by first decoding the D1 codewords, then the D2 codewords, and finally the D3 codewords. A non-interleaved approach transmits the packets in this same order — we send the packets which comprise the parallel D1 codewords, increment D2sn, send the packets which comprise the next set of parallel D1 codewords, increment again, and continue (see Figure 3). Consider encoding the message in the same order but sending the data across the channel by reordering the dimensions. Specifically, send a packet, increment D3sn by 1, send the next, increment D3sn by 1 again and continue until we reach the end of the D3 dimension. Then increment D2sn by 1, reset D3sn, and continue. At the receiver, put the data back into the proper order (deinterleave) and proceed with the decoding as before. This effectively sends the data across the channel in the opposite dimensional order as the decoding.

Now consider these two scenarios when faced with a long burst erasure. Roughly speaking, the non-interleaved scheme leaves the burst erasure recovery solely to the final decoding dimension — recovery from 2 erasures in the D3 codewords. Since we transmit the packets in a non-interleaved order, we can recover approximately 2 entire 3-d sets (D0,D1,D2) of data, and the final decoding loop (dimension D3) does all of the work. This yields an approximate burst erasure protection of $2 \times$ LEN1 $\times$ LEN2 packets. Similarly, the interleaved scheme leaves the burst erasure recovery solely to the *first* decoding dimension — recovery from 2 erasures in the D1 codewords. Since we transmit the packets in the interleaved order, we can recover approximately 2 rows (D2) from each of the 3-d sets of data, and the first decoding loop (dimension D1) does all of the work. This yields an approximate burst erasure protection of $2 \times$ LEN2 $\times$ LEN3 packets. If we simply swap the sizes of the dimensions, keeping the size of the 4-d data set constant, the two schemes are approximately equivalent in terms of burst erasure protection.

This fact seems surprising, but as mentioned above, we can explain it. We rely on a single decoding dimension for the main erasure recovery in the case of a long burst erasure. We just change the workhorse dimension when we interleave in this way. To increase burst erasure protection, we could opt to interleave several 4-d data sets over the channel. However, there are many important issues associated with this. Specifically, we would like to interleave 4-d data sets destined for different receivers so that a single receiver does not need to store multiple data sets concurrently (this would increase the memory requirement on the receiver). This effectively reduces the data rate to each receiver while maintaining the same throughput across the channel. Thus, channel outages of a fixed time duration correspond to reduced data loss per user. Such a scheme requires control over the messages generated from different sources and destined for different receivers, which is outside the scope of application-level forward error correction. Alternately, to avoid requiring access to global data, we could opt to interleave 4-d data sets destined for a single receiver and thus reduce the erasure burst length per data set. Such a scheme is possible when a single source has multiple messages for a single receiver or, more likely, a single long message that must be segmented into multiple 4-d data sets (such as a large image file). As mentioned above, however, this would significantly increase the memory requirement at the receiver.

4.2 Multiplication and Division in a Galois Field

For the encoding and decoding procedure, we need to perform arithmetic operations in a finite (Galois) field. We use the Galois Field of 256 elements. In this field (and in any GF of characteristic 2) addition may be efficiently implemented by a bitwise XOR of their m-tuple representations.

The primitive polynomial we use to derive field multiplication in GF(256) comes from Peterson and Weldon [2, App C]. We use $p(x) = x^8 + x^7 + x^6 + x + 1$. The 256 field elements are uniquely represented as the polynomials of degree 7 with binary coefficients (elsewhere called the m-tuple representation). Field multiplication is then equivalent to polynomial multiplication modulo $p(x)$.

An equivalent representation of field elements is as a power of a primitive element α, where α is a root of the primitive polynomial $p(x)$. The 255 non-zero polynomials of degree 7 with binary coefficients are precisely the first 255 powers of α ($\alpha^0 = 1$ to α^{254}) reduced modulo $p(x)$. Field multiplication using the powers-of-α representation corresponds to modulo-255 addition of the exponents — $\alpha^i + \alpha^j = \alpha^{(i+j)_{\mathrm{mod}255}}$. In the interest of time, we first implemented multiplication this way by using lookup tables to convert between the polynomial representation and the powers-of-α representation. However, it is clear that using a 2-key lookup table generated during initialization (whose entry is the multiplication of the 2 keys in m-tuple representation) yields a significant time improvement, so we added a quick patch in the C code to replace multiplication methods. The former method may be implemented using two conversion tables (size on the order of the field size 256) and a single mod-255 addition for each field multiplication. The latter method requires a single table lookup (size on the order of the field size squared). Both methods would generate their tables during initialization. Using the 2-key lookup table is faster due to the single table lookup, and we suggest exploiting this method since approximately $\frac{1}{2}$ of the core computation is multiplication (for computing the syndromes). Division is implemented similarly, though used only when correcting codewords with 2 erasures; it requires another table of equal size to implement as a 2-key lookup table.

4.3 Choosing the Dimension Sizes

Note that the implementation choice $n_0 = 400$ is rather arbitrary. This 400-byte UDP packet size includes 3 bytes for sequence numbers. Also, the fixed per-packet network overhead is approximately 85 bytes (8 byte UDP header including optional checksum, 20 byte IP header, approximate 12 bytes for ATM AAL-5 layer [depends on multiplexing MPEG streams], and $5\lceil\frac{400}{48}\rceil$ bytes for the 48-byte ATM cell headers). This represents approximately 17.5% network overhead. Note that this overhead would be incurred by any error control scheme using a 400-byte packet and thus has not been a focal point in the design or discussion.

There are two other factors to consider in setting the packet size n_0. First, the length does not affect the power of the product code in terms of dropped packets, but it clearly affects the performance in terms of channel time. For the same channel outage time, decreasing n_0 obviously increases the number of packets dropped. Second, varying n_0 does not greatly affect the per-byte decoding speed since the product code is implemented in parallel over the packet bytes.

Next consider the product code dimensions. Note that all burst performance guarantees can be stated in terms of the dimension sizes (see Section 3.2) n_1, n_2. In general, fixing the product $n_1 n_2 = N$ results in a fixed deterministic burst erasure guarantee. All other things being equal, we could opt to maximize the net code rate by choosing $n_1 = n_2 = \sqrt{N}$. Using the Lagrange multiplier method for real-valued n_1, n_2, denote the Lagrangian by J and the multiplier by λ. Then

$$\begin{aligned}
J &= \left(\frac{n_1-2}{n_1}\right)\left(\frac{n_2-2}{n_2}\right) + \lambda\left(N - n_1 n_2\right) \\
&= \left(1-\frac{2}{n_1}\right)\left(1-\frac{2}{n_2}\right) + \lambda\left(N - n_1 n_2\right). \\
\frac{\partial J}{\partial n_1} &= \left(1-\frac{2}{n_2}\right)\left(\frac{2}{n_1^2}\right) - \lambda n_2, \\
\frac{\partial J}{\partial n_2} &= \left(1-\frac{2}{n_1}\right)\left(\frac{2}{n_2^2}\right) - \lambda n_1.
\end{aligned}$$

Setting both partials to zero and equating,

$$\begin{aligned}
\left(1-\frac{2}{n_1}\right)\left(\frac{2}{n_2}\right) &= \lambda n_1 n_2 \\
&= \left(1-\frac{2}{n_2}\right)\left(\frac{2}{n_1}\right).
\end{aligned}$$

Solving simultaneously for positive n_1, n_2, we find $n_1 = n_2$ is the only possible solution, which is clearly a maximum. Note that a maximum is guaranteed to exist since we are maximizing a continuous function over a compact space. The extension to integral values is intuitive.

Note that, though the burst erasure performance does not depend on n_3, the net code rate does. The net rate of the code, disregarding network overhead, is $\mathcal{R} = \left(\frac{n_0-3}{n_0}\right)\left(\frac{n_1-2}{n_1}\right)\left(\frac{n_2-2}{n_2}\right)\left(\frac{n_3-2}{n_3}\right)$ (recall we use 3 bytes for sequence numbers in every packet). We do not want to make n_3 too small. On the other hand, the computer memory requirement is directly proportional to n_3.

Finally note that, barring changing the code dimensions on a per-message basis (see below), the code dimension choices further affect the net code rate when addressing the finite size of messages.

Recall that every message necessarily must be padded to fit into an integral number of 4-d data sets. This effect is impossible to quantify without a reasonable traffic profile (a probabilistic distribution on message size).

4.4 File Size versus Code Format

There is nothing inherent to the logistics of the scheme requiring a fixed set of code dimensions for every message. Thus, theoretically, we are free to vary the dimensions of the 4-d data set to fit the finite message size with minimal padding (and thus reduce additional coding overhead).

There is nothing inherent to the implementation, either, which would prohibit varying the code dimensions. We could simply pass in the appropriate dimension sizes to the decoder. Note that our implementation of the decoder must allocate space at compile time (since multi-dimensional arrays are used and the code is implemented in C) and thus must take the maximum amount of memory regardless.

There are several effects to consider when addressing dynamically varying code dimensions. First, we cannot vary the packet size n_0 without affecting the performance of the queue manager. The queue manager will be designed with a fixed unit of work in mind — e.g., the UDP packet size. However, this does not mean that queue manager performance will suffer; however, we may sacrifice predictability. Second, varying n_0 allows the finest-grain 4-d data size adjustment (with the smallest effect on net code rate per unit change of the four dimensions, as long as we ignore network overhead). Note that when adjusting the code dimensions, we should consider the per-packet network overhead rather than just the net code rate $\mathcal{R}$ as defined above. Third, if we allow dynamically varying dimensions, we must transmit the dimension parameters along with the data. Recall that burst erasures may wipe out a large number of packets, so sending this information in a "setup" packet or in the first few packets is unacceptable — the parameters may have to be transmitted with every packet, corresponding to additional coding overhead (like the packet sequence numbers).

The padding overhead reduced by allowing dimension variability is, once again, impossible to quantify without a probabilistic distribution on the traffic size. Since traffic is completely uncharacterized at the moment, no attempt at quantification has been made.

4.5 Availability of Data

We have entirely skirted the implementation issue of data availability. We include 3 bytes per packet denoting the packet position in the 4-d data set, (D1sn,D2sn,D3sn). This requires 3 bytes of overhead per packet, but it allows us to assemble the data in order at the receiver. When implementing the preliminary (benchmarking) procedure, we assume that all information has arrived at the decoder and processing can begin. We start with an entire simulated data set (with missing or dirty packets, as appropriate) and place this "received data" in order in RAM and proceed with the full decoding procedure outlined above.

On the other hand, this scheme was designed to allow efficient incremental decoding. This means that all computation based on an arriving packet can proceed when the packet arrives and that no further data is necessary before the computation on the packet can be completed. In an actual implementation, we envision an interrupt-driven decoding procedure (a per-packet interrupt would be excessively slow if data were arriving with any significant regularity, while only a single interrupt may be necessary if data were arriving at, say, 10 Mbits/sec).

There is an open implementation issue to consider — when should we give up waiting for data to arrive? Specifically, our implementation assumes we've already given up waiting for any further packets to arrive (we consider them erased). However, the absence of an expected packet (at a certain time) may correspond to a channel outage, a network checksum failure, network buffer overflow, a queue manager decision to service another user, sudden heavy traffic network delays (on a terrestrial network), etc. The first three result in a dropped packet, while the last two simply delay its arrival. Consider the latter case, wherein a packet is delayed somewhere in the network. Deciding to give up on an expected packet may make the difference between a successfully and an unsuccessfully decoded message (obviously). However, it may also make the difference between low and high decoding latency (when the decoder could recover the delayed packet by considering it "erased").

We also realize that packets may arrive out of sequence. Though we believe this should not happen in the network architecture we currently envision, it may be possible. Consider an application sending packets over multiple physical links in a terrestrial network. There is no guarantee that all packets will arrive at the queue manager (or from the satellite link receiver to the application) in sequence. In the current C code implementation, we effectively assume that all packets arrive in sequence. Jumping forward in the sequence is no problem in that we can simply assume all intermediate packets have been erased (and thus we do our best to fill in the intermediate packets). However, jumping back in the sequence of packet arrivals is impossible in the current implementation. We process syndromes for D1 (fixing D2sn and D3sn), correct the D1 codeword if possible, and then overwrite the D1 syndrome table with the next D1 codeword (for D2sn + 1 and D3sn). Similarly, we process syndromes for D2 (fixing D3sn), correct the D2 codewords if possible, and then overwrite the D2 syndrome table (for D3sn + 1). Though memory efficient, we cannot do this if we allow packets to arrive out of sequence.

5. SUMMARY

We have outlined a high-rate forward error control scheme designed to recover from a variety of network and channel behaviors which result in a significant loss of data. The scheme is designed for the protection of large messages destined for a receiver running an application-level message decoding procedure. The error control scheme has been designed for flexibility according to receiver computational and memory limitations as well as adaptability to channel behavior.

The packet loss patterns which will cause the proposed sub-optimal decoding procedure to fail have been fully characterized, and these patterns have been compared to those which would cause an optimal decoding procedure to fail. The deterministic burst packet loss protection in the presence of scattered packet erasures and packet errors has been characterized in a number of ways. A stochastic analysis has not been attempted because no reasonable stochastic channel model is available for the types of error events considered.

An encoding and decoding procedure was implemented in C for preliminary benchmark testing. We find that the full core decoding procedure can process incoming data on a Sun Sparc 10 and a Sun Ultra 1 with a reasonable throughput at the application level.

Several minor implementation issues have been highlighted which still need to be addressed if this proposed scheme is to be implemented operationally.

REFERENCES

[1] Richard E. Blahut. *Theory and Practice of Error Control Codes.* Addison-Wesley, Reading, MA, first edition, 1983.

[2] W. Wesley Peterson and E.J. Weldon Jr. *Error-Correcting Codes.* MIT Press, Cambridge, MA, second edition, 1972.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE 22 July 1997	3. REPORT TYPE AND DATES COVERED Technical Report

4. TITLE AND SUBTITLE A Forward Error Control Scheme For GBS and BADD	5. FUNDING NUMBERS C — F19628-95-C-0002 PR — 602
6. AUTHOR(S) Brett E. Schein Steven L. Bernstein	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Lincoln Laboratory, MIT 244 Wood Street Lexington, MA 02173-9108	8. PERFORMING ORGANIZATION REPORT NUMBER TR-1039
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA/ISO 2701 North Fairfax Drive Arlington, VA 22203-1714	10. SPONSORING/MONITORING AGENCY REPORT NUMBER ESC-TR-97-052

11.SUPPLEMENTARY NOTES

None

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This document provides a description of an error control scheme which can enhance the reliability of file and message transfer in the Battlefield Awareness and Data Dissemination (BADD) and Global Broadcast Service (GBS) programs.

The proposed scheme is intended to be general enough for adaptation to real network considerations, traffic profile, channel behavior, and computational and memory limitations. As such, the proposed scheme will need to be modified slightly for actual use, and some of the necessary modifications will be addressed.

14. SUBJECT TERMS	15. NUMBER OF PAGES 36
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Same as Report	19. SECURITY CLASSIFICATION OF ABSTRACT Same as Report	20. LIMITATION OF ABSTRACT Same as Report

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by AMSI Std. 239-18
298-102