Reliable Information Storage in Memories Designed from Unreliable Components*

By MICHAEL G. TAYLOR

(Manuscript received April 10, 1968)

This is the first of two papers which consider the theoretical capabilities of computing systems designed from unreliable components. This paper discusses the capabilities of memories; the second paper discusses the capabilities of entire computing systems. Both present existence theorems analogous to the existence theorems of information theory. The fundamental result of information theory is that communication channels have a capacity, C, such that for all information rates less than C, arbitrarily reliable communication can be achieved. In analogy with this result, it is shown that each type of memory has an information storage capacity, $\mathcal{C}$, such that for all memory redundancies greater than $1/\mathcal{C}$ arbitrarily reliable information storage can be achieved. Since memory components malfunction in many different ways, two representative models for component malfunctions are considered. The first is based on the assumption that malfunctions of a particular component are statistically independent from one use to another. The second is based on the assumption that components fail permanently but that bad components are periodically replaced with good ones. In both cases, malfunctions in different components are assumed to be independent. For both models it is shown that there exist memories, constructed entirely from unreliable components of the assumed type, which have nonzero information storage capacities.

I. INTRODUCTION

The problem of designing systems which operate reliably even though their components are unreliable has been formulated in many different ways. In a typical formulation, one considers some particular

* This work, which is based on part of a doctoral thesis submitted to the Department of Electrical Engineering, M.I.T., September 1966, was supported by the National Aeronautics and Space Administration (Grant NsG-334).

system which performs a computation with a nonzero probability of error. The problem is to design some other "reliable" system which performs the same computation using the same types of components but with a smaller probability of error. In fact, the ultimate objective is to show that it is possible to design systems, using only unreliable components, which perform computations with an arbitrarily small probability of error. Unfortunately, there is no standard terminology for describing these systems; therefore, the following section introduces the terminology to be used throughout this paper.

1.1 *Definitions*

The computations performed by the computing systems are described in terms of *elementary operations* where an elementary operation is any Boolean function of two binary operands. There are sixteen different elementary operations, each one of which can be represented by a binary matrix of the type shown in Fig. 1. Typical elementary operations are AND, OR, and modulo-2 addition. The computing systems to be considered are constructed from *components* which are devices that either perform one elementary operation or store one binary digit. The *complexity* of a system is defined to be equal to the number of components within the system.

In an irredundant computing system, the *amount of computation* performed by the system equals the number of elementary operations which are executed. Corresponding to each irredundant computing

AND	0	1
0	0	0
1	0	1

Fig. 1 — Binary matrix for AND operation. There are $2^4 = 16$ ways of filling this table, each one of which describes one of the 16 allowed Boolean functions.

system, there are many redundant computing systems which perform equivalent computations. These redundant systems are more complex than the equivalent irredundant one but, hopefully, they are also more reliable. The amount of computation performed by any one of these redundant computing systems is defined to be equal to the amount of computation performed by the corresponding irredundant system. Finally, the *redundancy* of a computing system equals the ratio of the complexity of the system to the amount of computation performed by the system.

To illustrate the use of these terms, consider a system that computes, in parallel, the modulo-2 sums of the digits in two k-digit sequences. The system first encodes each sequence of digits into a code word from an (n, k)* group code, then forms the modulo-2 sums of the digits in these code words, and finally decodes the result. The *amount of computation* equals k since an equivalent irredundant computer would simply perform k *elementary operations* each consisting of a module-2 sum. If the *complexities* of the encoder and decoder within the redundant system are C_E and C_D, respectively, the *complexity* of the entire system equals $C_E + C_D + n$, where the last term arises from the n modulo-2 adders required to perform the desired operation. The *redundancy* of this system equals $(C_E + C_D + n)/k$.

1.2 *Historical Background*

Von Neumann was one of the first to propose a system which uses redundancy to gain reliability.[1] He considered systems consisting of interconnections of identical elements† where all the elements compute either the majority function or the Sheffer-stroke function. The form (network topology) of the redundancy network is similar to that of the original irredundant network, the precursor. Specifically, each element in the precursor is replaced by a set of $3n$ elements of the same type in the redundant network (redundancy $= 3n$), and each interconnection is replaced by a bundle of n interconnections. The $3n$ elements in each set are interconnected in such a way that there are n outputs. It is assumed that a malfunction occurs in a particular set of elements whenever more than a certain fraction, θ, of the n outputs are in error; θ is chosen to minimize the probability of a malfunction within the entire system. Von Neumann showed that, for large n, the

* n is the length of each code word; k is the number of information digits in each word.

† The terms "element" and "network element" are used to indicate devices consisting of several components (some finite number).

probability that one set of $3n$ Sheffer-stroke elements malfunctions on one particular use is

$$\Pr(\text{malfunction in one set of elts.}) \cong 6.4/(n)^{\frac{1}{2}} \cdot 10^{-8.6n/10,000}$$

where it is assumed that the probability of error for each use of each element is 5.10^{-3}. Therefore, for this system, the probability of malfunction decreases exponentially with the redundancy, provided that the redundancy is sufficiently large. Other approaches involving the use of more complex modules have led to results similar to those of von Neumann.[2] In some cases the resulting network is more efficient (less redundant for a given probability of system failure) than von Neumann's network. However, in all cases, to achieve an arbitrarily small probability of system failure it is necessary to make the redundancy arbitrarily large.

It is interesting to compare von Neumann's results with those obtained by Shannon concerning the reliability of communication systems.[3] Both show that the probability of error within the system can be made arbitrarily small. In the case of communication systems, this can be achieved for certain nonzero information rates by choosing the constraint length of the code arbitrarily large. The largest information rate for which the probability of error can be made arbitrarily small is called the *capacity of the communication channel.* By making an analogy with this result, one might expect that, in the case of a computing system, it should be possible to achieve an arbitrarily small probability of error for certain bounded values of the redundancy by choosing the complexity sufficiently large. Extending this analogy, the reciprocal of the minimum redundancy for which the probability of error can be made arbitrarily small is called the *computing capacity* of the computing system. For the systems proposed by von Neumann, there is no finite redundancy for which the probability of error can be made arbitrarily small; therefore, these systems have a computing capacity of zero.

The question remained whether there existed any method for designing a "reliable" system with a nonzero computing capacity. Since it was well known that by using suitable coders and decoders it is possible to obtain a "reliable" communication system, it was natural to attempt to apply coding techniques to the problem of designing a "reliable" computing system. One approach is to consider coding the inputs to each computing component and decoding the output. Elias set out to show that this method could not be used to design a general computing system with a nonzero computing

capacity.[4] Since Elias desired a negative result, he permitted all coders and decoders to be noiseless, and since a general computing system must be capable of performing all 16 elementary operations, Elias considered 16 types of computing components, each capable of performing one of the elementary operations.

The computing components were divided into two classes. The operations performed by components in the first class were those represented by matrices containing an odd number of ones and zeros and in the second class were those containing an even number. Elias showed that components in the first class have a computing capacity equal to zero; but that it is possible for components in the second class to have a nonzero computing capacity.*

Unfortunately, the only component in the second class which performs a nontrivial operation is the modulo-2 adder; furthermore, there is no combination of class two components that can perform class one operations such as AND and OR. Therefore Elias concluded that this coding technique could not be used to design a general computing system with a nonzero computing capacity.

More recently Winograd and Cowan proposed another scheme for designing a "reliable" computing system.[8] Their approach was very similar to von Neumann's. However, instead of considering a single irredundant network as the precursor for the redundant network, Winograd and Cowan considered a composite network consisting of k copies of this irredundant network, each computing independently, to be their precursor. If the original irredundant network has a complexity σ then this composite network has complexity $k\sigma$; but its redundancy is still one since each network is capable of performing independent operations on different inputs.

To introduce redundancy into this composite network, one considers sets of k network elements, the members of each set being the corresponding elements in each of the k precursors. The redundant network is formed by replacing each set of k elements with n modules. These modules have the property that each of their inputs is encoded according to some (n, k) block code, thus allowing each one to perform an error correction operation on each of its inputs. Every set of n modules performs the appropriate operations on the corrected inputs so that the n binary outputs from the n modules form the

* To achieve this nonzero computing capacity, it is necessary to assume that the complexity of the encoders and decoders grows only linearly with the block length of the code as in the case of convolutional coders and sequential decoders.[5-7]

code word corresponding to the desired result, namely the code word whose information digits are equal to the corresponding k outputs in the original composite network.

Winograd and Cowan assumed that modular malfunctions are statistically independent from one module to another; furthermore, they also assumed that for all modules except the "output modules" the probability of a modular malfunction is p_0, independent of the operations performed by that module. The "output modules," those modules whose output constitutes the output of the computing system, were assumed to be noiseless. To compute a bound on the probability of error for this system, each set of n noisy modules can be modeled by a set of n noiseless modules, where the output from each module is passed through a binary symmetric channel (BSC) with crossover probability p_0. A failure occurs whenever the output from any set of n BSC's is such that a noiseless decoder would be unable to decode this word correctly. According to the noisy channel coding theorem,[9] there exist codes for which the probability of such a failure is bounded by

$$\Pr\,[\text{modular failure}] \leqq e^{-nE(R)}$$

where $E(R) > 0$ for codes with information rates less than the capacity of a BSC with crossover probability p_0. Since there are at most $n\sigma$ modules within the network and since each module is used only once during the computation, the probability of a malfunction anywhere within the network during the computation is bounded by

$$\Pr\,[\text{failure in network}] \leqq n\sigma \cdot e^{-nE(R)}$$

which can be made arbitrarily small by making n sufficiently large.

If the complexity of the modules were fixed, this result would imply that the probability of error can be made arbitrarily small by making the complexity sufficiently large, while keeping the redundancy bounded. Unfortunately, each module must perform encoding and decoding which requires a number of operations that grows at least linearly with n. Therefore, the complexity of the modules must grow at least linearly with n which implies that the redundancy of the overall network must grow at least linearly with n rather than being bounded as one would have hoped. Therefore, the probability of error for the system can be made to approach zero only in the limit of infinite redundancy. Hence, Winograd and Cowan's system also has a computing capacity equal to zero.

1.3 *Error Criteria*

Although the term "probability of error" is used in connection with each of the three systems discussed in the previous section, the error criterion was different for the different systems. Elias, Winograd, and Cowan assumed that each input to the computing system is uncoded and that the output from the system is also uncoded. They assumed that for each set of inputs, the system has one correct output defined as the output which would be obtained if the system were noiseless. If the actual output differs from the correct output, the system has made an error. To obtain a probability of error for the system that is not lower bounded by the probability of error for the output components, they required that the output components be noiseless.

The necessity for using noiseless output devices arises because of the requirement that the output be uncoded. Von Neumann avoided this problem by assuming that all inputs and outputs are repeated n times. He assumed that the result is "correct" provided that the fraction of the outputs which are in error is small. Von Neumann's assumption that all inputs and outputs must be repeated is a special case of the assumption that all inputs and outputs must be coded according to some error correcting code. The latter, more general assumption is made in this paper. The only condition that is imposed is that both the inputs and the output must be coded according to the same code so that computing systems are compatible with each other. Since the outputs are coded, there must exist classes of outputs corresponding to the decoding equivalence classes of the code. A result is considered to be correct provided that it is within the class which contains the code word corresponding to the desired result.

The concept of coded inputs and outputs might, at first, seem unrealistic since it implies that the user is capable of performing error correcting coding and decoding. However, if we consider the case of two people communicating with each other, we observe that a very complicated process of coding and decoding takes place. Appropriate redundancy is introduced not only through the inherent redundancy in the language but also through the use of "diversity channels" which, in this case, correspond to facial expressions, hand and arm movements, voice inflections, and so on. Therefore, since there is always an appropriate coding used in the transmission of information between individuals, it is unrealistic to expect that a machine and user could communicate without the use of some type of

error correcting procedure. In fact, the condition that all information must be coded is a necessary requirement for the reliability of a computing system in which every component is noisy.

1.4 *Synopsis*

It is our ultimate objective to show that it is possible to construct from unreliable components a reliable computing system where a *reliable system* is defined as a type of system which has a nonzero computing capacity. For the first part of the analysis, it is assumed that component errors are statistically independent from one component to another and from one use of a particular component to another use. A computing system constructed from components of this type is called a *noisy computing system.* A virtually identical analysis and similar results apply in the case where components within the system fail permanently but where periodic maintenance is performed on the system; that is, at regularly spaced times, components which have failed are replaced by good ones. In this paper we restrict our attention to memories. It is shown that information can be stored reliably within a "stable memory," a device constructed entirely from unreliable components. The paper following this shows that it is possible to design a computing system, using unreliable components, that performs operations reliably on information stored within stable memories.

II. STABILITY

The remainder of this paper is concerned with reliable information storage in memories constructed from unreliable components. A memory is a device in which information is stored at one time and recovered at some later time. If a memory is to be useful, it must have two important characteristics. First, it should be possible both to store information in the memory and to read information from the memory at any time specified by the user, or at least at any one of a set of discrete times where the members of the set are closely spaced. All memories considered in this paper can have information stored in them at any time and retrieved from them at any time. Second, the information read out of the memory must be identical to or at least equivalent to the information originally stored. With memories constructed entirely from unreliable components, it is unreasonable to expect that the word read out of the memory will be identical to the word stored; however, we can hope that the informa-

tion will be preserved. To clarify this idea of information preservation we introduce the concept of stability.

2.1 *The Concept*

To illustrate the meaning of stability, let us consider a simple memory consisting of one noisy register and a correcting network. The *state* of this memory is defined by the word contained within the register. It is assumed that initially ($t = 0$) a code word from some error correcting code is read into the register, thus defining the memory's initial state. Since the register is noisy, errors occur in the stored digits; hence, the state of the memory is perturbed away from the original one. The correcting network monitors the contents of the register, performs error correction, and inserts into the register an estimate of the original code word. If there were no correcting network, the state of the memory would "wander away" from the original one; however, the correcting network provides a "restoring force" which tends to bring the state of the device back to the original. The noise may perturb the state of the device beyond the error correcting capability of the correcting network. If this happens, we say that a *memory failure* has occurred. To define a memory failure more precisely, it is necessary to associate with the different input code words disjoint classes of states. As long as the state of the memory remains within the appropriate class, no memory failure has occurred.

The redundancy of a memory is defined to be the ratio of the complexity of the memory to the complexity of an irredundant memory which has the same information storage capability. The inputs to the memory being considered are code words from an (n, k) block code; hence, the memory has a storage capability of k bits. This memory is denoted by M_k. An irredundant memory with the same information storage capability as M_k would have a complexity equal to k, since it would consist of k one-bit information storage components. If we consider k to be a variable, we can speak of a sequence of memories where M_k is a typical member of the sequence. If the complexity of every memory in this sequence is less than $\alpha \cdot k$, the redundancy of any memory in the sequence must be less than α which, by assumption, is independent of k. Hopefully, for any T, it is possible to make the probability of a memory failure during the time interval $0 \leqq t \leqq T$ arbitrarily small by choosing k sufficiently large while keeping α bounded. If the sequence of memories has this property, we say that the sequence is *stable*. For convenience we refer to a typical member of a stable sequence of memories as a *stable memory*. Here is a more concise definition of stability.

2.2 *The Definition*

Consider a sequence of memories denoted by $\{M_i\}$. The memories in this sequence are ordered according to their information storage capability; that is, the kth memory, M_k, has a storage capability of k bits. The sequence $\{M_i\}$ is called *stable* if it satisfies the following conditions:

(*i*) For any k, M_k must have 2^k allowed inputs which are denoted by $\{I_{ki}\}$, $0 < i \leqq 2^k$.

(*ii*) With each input there must be associated a class of states of M_k. The classes associated with different inputs must be disjoint. For any k and i, the class of states corresponding to I_{ki} is denoted by $C(I_{ki})$.

(*iii*) The complexity of M_k must be bounded by $\alpha \cdot k$ where α is a fixed parameter for any particular sequence.

(*iv*) Suppose that at $t = 0$ any one of the allowed inputs is transmitted to each memory in the sequence. Let there be no inputs for all $t > 0$. Consider a typical memory M_k. Denote the particular input that was transmitted to M_k by I_{ki}. The probability that the state of M_k does not belong to $C(I_{ki})$ at $t = T$ is denoted by $p_{ki}(T)$ and $\max_i [p_{ki}(T)]$ is denoted by $p_k(T)$. If the sequence is stable then, for any $T > 0$ and $\delta > 0$, there must be a k such that $p_k(T) < \delta$.

2.3 *Examples of Memories*

To further clarify the meaning of stability, consider two types of memories. The first consists of a noisy register without any correcting network and the second consists of a noisy register with a noiseless correcting network. In light of the discussion in Section 2.1, we do not expect that memories of the first type can be stable whereas we do expect that memories of the second type can be. These expectations are correct.

Consider first a sequence of noisy registers. A typical member of this sequence is a register containing n binary digits which define the register's state. Let p denote the probability that any particular digit stored in the register is changed during a time interval τ. If one is interested in the contents of the register only at the times $t = 0, \tau, 2\tau, 3\tau, \cdots$, a model for the noisy register is the noiseless register together with the n binary symmetric channels shown in Fig. 2. The allowed inputs to this noisy register, $\mathfrak{M}_k$, are the 2^k code words from some (n, k) code. The class of states corresponding to a particular input is the decoding equivalence class corresponding to that code word. The complexity

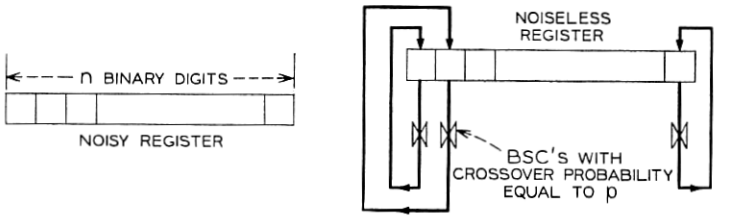


Fig. 2 — Model for a noisy register. The digits in the noiseless register are transmitted through the BSC's once every τ seconds.

of $\mathfrak{M}_k$ is $n = k/R$, where R is the information rate of the code; therefore, the appropriate value of α for this sequence is $1/R$.

Suppose that at $t = 0$ one of the allowed inputs is transmitted to $\mathfrak{M}_k$. At $t = \tau$ the probability of error per digit is p. The probability of a memory failure at $t = \tau$ equals the probability that the word in the register at $t = \tau$ does not belong to the decoding equivalence class containing the original input. Provided that the code is at least as "good" as an average random code, the noisy channel coding theorem[9] states that $p_k(\tau)$ is bounded by

$$p_k(\tau) \leqq \exp - [(k/R)\ E(R)]$$

where $E(R)$ is positive for $R < 1 - H(p)$. $[1 - H(p)]$ is the capacity of a BSC with crossover probability p. Since the probability of failure is independent of the particular input it is unnecessary to perform the maximization over all inputs.

Next consider the state of the register at time $t = T = L\tau$. According to the model in Fig. 2, to determine the state of the noisy register at $t = L\tau$ we must transmit the n binary digits through their respective BSC's L times. This is equivalent to transmitting each binary digit through L BSC's in series. Since the overall capacity of these channels in series decreases asymptotically to zero as L increases, for any fixed rate there must exist some L for which the capacity of the L channels in series is less than R. Therefore, for any sequence of noisy registers with bounded redundancy (fixed α or R), there must exist some L (or T) such that there is no register in the sequence which has a sufficiently small probability of failure to satisfy the requirements for stability. Thus, as would be expected, noisy registers are not stable.

As a second example, consider the same sequence of noisy registers but this time associate with each register a noiseless correcting net-

work. Within each time interval of length τ, this correcting network takes the output from the n BSC's and maps this vector onto a code word which is then inserted into the register to define its new state. This type of device is shown in Fig. 3. The operation performed by the correcting network is equivalent to that performed by a noiseless decoder followed by a noiseless encoder. This operation can be performed by a correcting network whose complexity is proportional to k; for example, a noiseless sequential decoder followed by a noiseless convolutional encoder.[5-7] If such a correcting network is used, the redundancy is independent of k and therefore can be bounded for all k. The probability of a memory failure at $t = \tau$ is again upper bounded by $\exp - [(k/R)\ E(R)]$ but the probability of a memory failure at $t = L\tau$ is now bounded by

$$p_k(L\tau) < L \cdot \exp - [(k/R)\ E(R)]$$

according to the union bound. Therefore, for any finite L, $p_k(L\tau)$ approaches zero as k approaches infinity provided that $R < 1 - H(p)$. This shows that a noisy register with a noiseless correcting network can be stable.

III. THE STABILITY OF MEMORIES CONSTRUCTED ENTIRELY FROM UNRELIABLE COMPONENTS

We now show that a noisy register with a noisy correcting network can be stable. This proof of stability is extended to memories in which components fail permanently but where the components which have failed are periodically replaced.

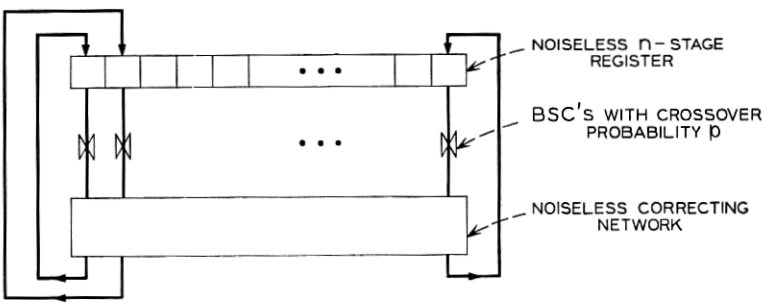


Fig. 3 — Noisy register with noiseless correcting network.

3.1 *The Importance of Low-Density Parity-Check Codes*

The memories to be considered store information in the form of code words from an error correcting code. Each memory consists of one or more noisy registers and a noisy correcting network that performs operations which are very similar to those performed by a decoder. It is our objective to show that noisy memories of this type can be stable. Since the complexity of a stable memory is required to be bounded by $\alpha \cdot k$, where k is the information storage capability and α is a proportionality factor which does not depend on k, the complexity of the correcting network in any stable memory can grow only linearly with k. There are only two kinds of correcting networks (decoders) known to have this property. One is a correcting network based on a sequential decoder[6, 7] and the other is a correcting network based on a low-density parity-check decoder.[10]

In deciding whether a particular correcting (decoding) procedure is suitable for use in a noisy correcting network, one must consider whether there are any essential steps in the procedure which could not be performed with a small probability of error by a noisy device. For example, almost all parity-check decoders are required to compute the modulo-2 sum of a set of digits where the number of digits in the set is proportional to the constraint length, N, of the code.

To compute the probability that a noisy device makes an error in performing this operation, consider the noisy addition network shown in Fig. 4. This network, consisting of $N - 1$ adders (modulo-2), computes the modulo-2 sum of N binary inputs. Let us assume that each adder in the network has a probability of error p_a and that adder errors are statistically independent from one adder to another and from one use of a particular adder to another. The output of the noisy addition network will be in error if an odd number of adders make errors. It can be shown[10] that the probability of this event equals

$$\text{Prob of error} = \frac{1 - (1 - 2p_a)^{N-1}}{2}.$$

This probability approaches $\frac{1}{2}$, exponentially with N, as N approaches infinity.

The memories under consideration store coded information. To make the probability of a memory failure arbitrarily small (as required for stability), one would expect that it would be necessary to make the constraint length of the code arbitrarily large. Therefore, the noisy correcting network must be able to perform error correction

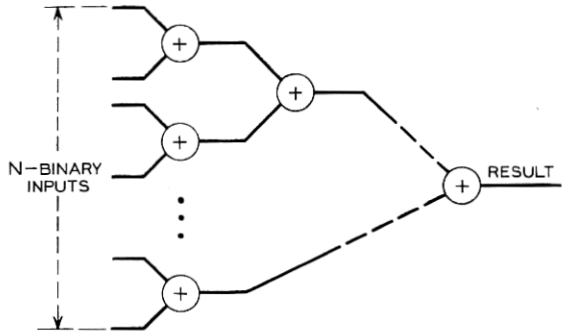


Fig. 4 — Modulo-2 addition network. ⊕ represents one adder (modulo-2).

even if the constraint length of the code is very long. This implies that no correcting procedure involving a modulo-2 addition operation of the type just described is suitable for use in a noisy correcting network. For example, the correcting network based on a sequential decoder must generate hypothesized branches of a code tree. Each digit on one of these branches is computed by forming the modulo-2 sum of a set of digits where the number of digits in the set is proportional to the constraint length of the code. The probability that a noiseless sequential decoder makes an error can be made arbitrarily small only if this constraint length is made arbitrarily large. But making the constraint length arbitrarily large makes the probability of an addition error within the noisy decoder arbitrarily close to ½. Therefore, a noisy correcting network should not be based on a sequential decoder.

Fortunately the correcting network based on a low-density parity-check decoder does not have this problem. A low-density parity-check decoder does evaluate parity checks, modulo-2 sums of the digits in parity-check sets; however, the number of digits in each parity-check set is not a function of the block length of the code.

3.2 *The Correcting Algorithm*

The memories to be considered consist of several registers and a correcting network and they store information in the form of code words from a low-density parity-check code. It is our objective to show that memories of this type can be stable. The first step is to consider the details of the correcting algorithm. This requires a brief explanation of low-density parity-check codes.

An (N, J, K) low-density parity-check code is defined to be a code with block length N such that there are K digits in each parity-check set and J parity-check sets containing any particular digit. Such a code can be represented by a parity-check matrix which has K ones in each row and J ones in each column. For example, an $(N = 20, J = 4, K = 5)$* low-density parity-check matrix is shown in Fig. 5.

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix}$$

$$d_{21}\ d_{31}\ d_{41}\ d_1\ d_{42}\ d_{22}\ d_{32}\ d_{43}\ d_2\ d_{33}\ d_0\ d_{11}\ d_{12}\ d_{13}\ d_{14}\ d_{23}\ d_{34}\ d_{44}\ d_{24}\ d_3$$

Fig. 5 — The parity-check matrix for a (20, 4, 5) low-density parity-check code. The digit positions, denoted by d with appropriate subscripts, are numbered in the way described in Fig. 6.

Gallager has described two schemes for decoding low-density parity-check codes, both of which are iterative.[10] However, we shall only be concerned with the simpler one. Each iteration of this scheme consists of first computing all the parity checks and then changing the value of any digit that is contained in more than a certain fixed number of unsatisfied parity-check constraints (if a parity check equals one, the corresponding parity-check constraint is unsatisfied). Provided that there were not too many errors initially, each successive iteration decreases the number of digits in error and, eventually, all parity check constraints are satisfied indicating that the resulting word is a code word.

To illustrate how this method works, let us suppose that the digit d_0 is in error but that all other digits are correct. In this case, all parity-check constraints involving d_0 will be violated, whereas at most one parity-check constraint involving any other digit will be violated. Therefore, d_0 will be changed whereas all other digits will be unchanged. In this way the digit d_0 will be corrected. If there are

* Notice that K is always greater than J.

errors among the digits used to check d_0, this digit may not be corrected during the first iteration. However, after one or more iterations sufficiently many of these errors may have been corrected to allow d_0 to be corrected also.

This simple decoding algorithm does have one problem. Recall that for any digit d_0, the original values of all the other digits in the J parity-check sets containing d_0 are involved in the determination of the new value of d_0 and similarly that the original value of d_0 is used in computing the new value of each digit in these J parity-check sets. This means that on successive iterations the values of the digits involved in making a new estimate of d_0 depend on the previous estimate of d_0. This leads to a complex interrelation between the errors which degrades the decoder's performance and, needless to say, greatly complicates its analysis. Fortunately Gallager has suggested a way to modify the algorithm which at least partially solves this problem. Using this modified algorithm, J estimates of each digit are made, each one being computed using a different combination of $J - 1$ out of the J parity-check sets containing the digit to be estimated. The value of a particular estimate is changed if $J/2$ or more out of the $J - 1$ parity checks are equal to one.

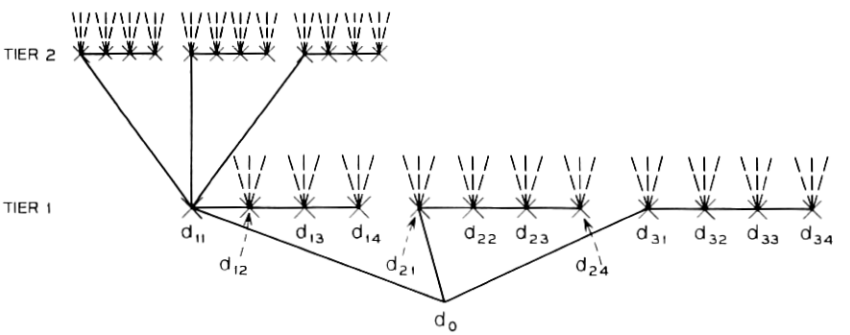
To construct a correcting network based on this algorithm, one starts with J registers of length N. Although the assignment of digit estimates to register locations can be arbitrary, one would probably choose to assign one estimate of each of the N digits in a code word to the corresponding N locations in each register. Since each estimate of a digit, say d_0, is to be made on the basis of $J - 1$ parity-check sets, each containing $K - 1$ digits other than d_0, there must be $(J - 1) \cdot (K - 1)$ other digits interconnected with the input to d_0. Using the modified algorithm, it is necessary to specify not only the digits to be interconnected but also the appropriate estimate of each digit. For example, consider the parity check set $(d_0, d_{11}, d_{12}, d_{13}, d_{14})$. The appropriate estimate of the digit d_{11} to be used in correcting d_0 is that estimate based on the $J - 1$ parity check sets which omit the one containing d_0. This estimate is used so that the values of the digits involved in computing the new estimate of d_0 do not, themselves, depend on a previous estimate of d_0.

This correcting network performs many iterations. During the first iteration each digit is estimated on the basis of the $(J - 1) \cdot (K - 1)$ digits to which it is interconnected. Since, during the second iteration, the same operations are performed, the resulting second estimate of each digit depends on the first estimates of these $(J - 1)$

$\cdot(K - 1)$ interconnected digits which, in turn, depend on the initial estimates of a much larger set of digits. The sets of digits involved in making successive estimates of some digit, say d_0, can be represented by means of parity-check set trees of the type shown in Fig. 6. The branches rising from d_0 represent one set of $J - 1$ parity-checks containing this digit. The interconnected nodes on the first tier of this tree represent the digits, other than d_0, in one of these parity-check sets. Each digit on the first tier of the tree is also contained in $J - 1$ other parity-check sets. These other parity-check sets are represented by the branches rising from the first tier to the second tier of the tree. This parity-check set tree can be extended indefinitely. The structure of the tree, beyond the first tier, is completely specified by the parity-check sets of the codes which, in turn, are specified by the parity-check matrix.

To see how the tree represents the digits involved in estimating d_0, let us suppose that the decoder has performed i iterations. During each iteration the digits on each tier of the tree are used to estimate the digits on the tier immediately below. Hence, after i iterations, the value of each digit, particularly d_0, has been influenced by the values of the digits on the first i tiers above it.

If a parity-check set tree is extended for many tiers, eventually some digit will appear in two different places in the tree. If the first repetition within any of the $J \cdot N$ trees occurs on the $(m + 1)$th tier,



$$
\begin{aligned}
d_0 + d_{11} + d_{12} + d_{13} + d_{14} &= c_1 \\
d_0 + d_{21} + d_{22} + d_{23} + d_{24} &= c_2 \\
d_0 + d_{31} + d_{32} + d_{33} + d_{34} &= c_3 \\
d_0 + d_{41} + d_{42} + d_{43} + d_{44} &= c_4
\end{aligned}
$$

Fig. 6 — Two representations for the parity-check constraints involving d_0. At the top is one of the J parity-check set trees rising from d_0. Beneath it are the parity-check equations containing d_0.

then, according to Gallager's nomenclature, the code has m *independent iterations*. Notice that this number, m, is a parameter of the code which is unrelated either to the statistics of the noise or to the particular decoding algorithm. This number plays a particularly important part in the analysis of the correcting procedure as it is shown that errors in specific sets of digits are statistically independent during these first m iterations.

The physical configuration of the memory is shown in Fig. 7. Within the correcting network there are $J \cdot N$ identical sets of components. Each set of components performs the operations required to estimate one particular digit. Let us consider a set of components which computes estimates of the digit d_0. The first operation that must be performed by this set of components, the computation of the $J-1$ parity checks rising from d_0, requires $(J-1) \cdot (K-1)$ two-input binary modulo-2 adders. The second operation, deciding whether the digit d_0 should be changed, can be performed by a decision device (threshold device) the output of which is a 1 if d_0 is to be changed and a 0 if d_0 is not to be changed. Finally, the output of this decision device must be added modulo-2 to the previous estimate of d_0, the operation requiring one binary adder (modulo-2). Similar operations are performed to obtain estimates of all $J \cdot N$ digits. These operations

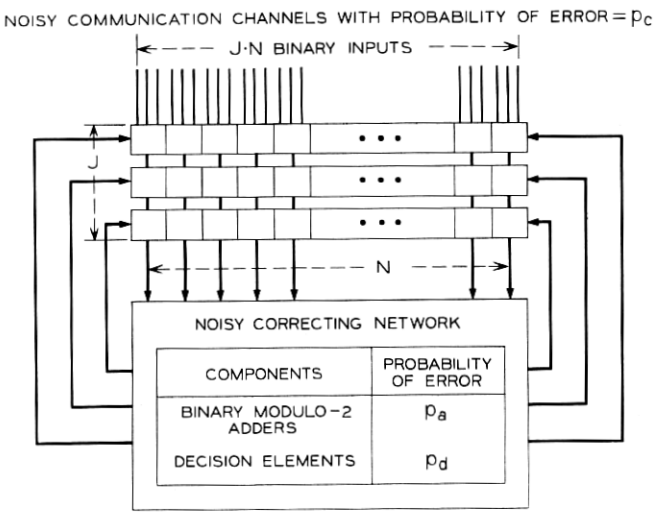


Fig. 7 — Physical configuration of a memory based on a low-density parity-check decoder.

constitute one correcting cycle (iteration). The correcting network performs correcting cycles continuously once information has been read into the registers.

Finally, let us consider the situations which cause the estimate of the digit d_0 to be in error after a particular iteration. Since any digit d_0 which is in error will be corrected only if a sufficient number of the parity checks containing d_0 are equal to 1, ideally, we would like every parity check containing d_0 to equal 1 if d_0 is in error but to equal 0 if d_0 is correct. If a parity check does not equal the desired value we say that the parity check is in error. Thus a parity check error depends on errors made by the adders and errors in the digits involved in estimating d_0, but not on the value of d_0 itself.

To simplify the mathematical analysis, we restrict our attention to the class of low-density parity-check codes with $J = 2l$, $l = 2, 3, 4, \ldots$ and the correcting algorithm stated previously. A set of events each one of which alone leads to an error (indicated by ϵ) in the estimate of the digit d_0 after a particular iteration are:

(*i*) $d_0 = \epsilon$ after previous iteration and $J/2$ or more parity checks are in error.

(*ii*) $d_0 \neq \epsilon$ after previous iteration and $J/2$ or more parity checks are in error.

(*iii*) The decision (threshold) device makes an error.

The first two conditions demonstrate an interesting property of this class of low-density parity-check codes. For this class of codes the conditions leading to an error in any digit after any iteration are the same whether or not the digit was in error before the iteration. This property will help to simplify the following mathematical analysis.

3.3 *The Stability Theorem for Noisy Memories*

Theorem 1: There is a stable sequence of noisy memories where every component in every memory has a fixed, nonzero probability of error per use.

Proof: The memories under consideration consist of J registers of length N, a noisy correcting network based on a low-density parity-check decoder, and a set of communication channels over which the inputs are transmitted. The definition of stability given in Section 2.2 includes specific conditions that must be satisfied by stable memories. To prove that the memories under consideration are stable, we must show that they satisfy all these conditions.

The definition of stability requires that there be a set of allowed

inputs for each memory. Each allowed input for a memory of the type under consideration consists of J copies of a code word from some (N, J, K) low-density parity-check code. If R equals the information rate of the code, the memory having registers of length N has $2^{RN} \triangleq 2^k$ allowed inputs. This memory is denoted by M_k. To define a sequence of memories, we allow k(and N) to be a variable but keep the parameters of the code, J and K, fixed.* Let the 2^k allowed inputs to M_k be denoted by $\{I_{ki}\}$ $0 < i \leqq 2^k$. With each input we must associate a class of states of M_k. The classes to be used are essentially the decoding equivalence classes of the code. To be more precise, the classes can be described in terms of the equilibrium states of M_k, denoted by $\{E_{ki}\}$ $0 < i \leqq 2^k$, where an equilibrium state is one in which every register contains one and the same code word. With each input, I_{ki}, there is a corresponding equilibrium state, E_{ki}, in which the registers contain the code words represented by I_{ki}. The state of M_k belongs to the class $C(I_{ki})$ if a noiseless correcting network (that is, a noiseless low-density parity-check decoder) could correct all the errors; in other words, if its final state would be the equilibrium state, E_{ki}.

The definition of stability requires that for each sequence of memories there must be an α such that for every k the complexity of M_k is bounded by $\alpha \cdot k$. The complexity of the correcting network in M_k is computed in Section 3.2. If each of the J registers in M_k is of length N, the correcting network must contain $[1 + (J-1)(K-1)]\, J \cdot N$ binary adders (modulo-2) and $J \cdot N$ decision devices. Since a decision device must determine whether $J/2$ or more out of the $J-1$ parity checks are in error, the complexity of each decision device depends only on the code parameter, J, which is fixed for any particular sequence of memories. Thus the complexity of each decision device, denoted by D, is independent of k. Finally, the noisy registers within the memory must contain $J \cdot N$ storage components. Therefore the complexity of M_k is:

$$\begin{aligned} \text{Complexity of } M_k &= [2 + D + (J-1)(K-1)]J \cdot N \\ &= \frac{[2 + D + (J-1)(K-1)]J}{R} \cdot k. \end{aligned}$$

Since Gallager[10] has shown that $R \geqq 1 - J/K$, the complexity proportionality factor for the sequence of memories under consideration is upper bounded by

* There are low-density parity-check codes for all values of N which are integer multiples of K.[10]

$$\alpha \leqq \frac{[2 + D + (J - 1)(K - 1)]J}{1 - J/K}.$$

Every component has a nonzero probability of making an error. The error probabilities are denoted:

p_r = probability that one particular binary digit within the register changes during a time interval τ, the time required for one correcting cycle.
p_a = probability that an adder makes an error on any one use.
p_d = probability that a decision device makes an error on any one use.
p_c = probability of an error in transmitting a digit across a communication channel.

It is assumed that errors are statistically independent from one component to another and from one use of a particular component to another use.

To prove that the information storage devices in question are stable, it must be shown that for any $\mathcal{L}$, the probability of a memory failure in M_k during the time interval $0 \leqq t \leqq \mathcal{L}\tau$ can be made arbitrarily small by choosing k sufficiently large. In the remainder of this section, the probability of a memory failure for the typical device, M_k, is upper bounded. To determine whether a memory failure has occurred at some particular time we use the following algorithm.

Imagine that M_k becomes noiseless at that time and that the noiseless correcting network within M_k performs m more iterations where m is the "number of independent iterations" described in Section 3.2. These are precisely the same operations that a noiseless low-density parity-check decoder would perform. If the final state of the hypothetical noiseless memory is error free (that is, equals the equilibrium state corresponding to the original input) then, by definition, no memory failure occurred. Thus, to bound the probability of a memory failure we must bound the probability that the final error pattern is not error free.

In general, the error pattern within the registers of M_k depends on all the component errors that have occurred since the original input was transmitted to the memory. However, if certain conditions are satisfied, the error pattern is a function of only the component errors that occurred during the most recent m iterations. If no component errors occur during these m iterations and if the required conditions are satisfied, the final error pattern will be error free. On the other hand, if these conditions are not satisfied we say that a *propagation*

failure has occurred. Therefore, to bound (upper bound) the probability of a memory failure, we bound the probability that a propagation failure occurs on the mth noiseless iteration of the hypothetical noiseless memory.

The first step is to show that, in the absence of a propagation failure, the probability of error for digits stored in the memory (the probability of error per digit) has an upper bound, denoted by p_0, such that $p_0 \ll \frac{1}{2}$. This can be seen intuitively by comparing the performance of a noisy corrector with that of a noiseless corrector, that is, a noiseless low-density parity-check decoder. Provided that the initial probability of error is not too large, a noiseless corrector decreases the probability of error per digit with each iteration until this probability reaches zero. If the probability of error for the components within the noisy corrector is small, compared with the initial probability of error per digit, for the first few iterations the noisy corrector decreases the probability of error per digit just as the noiseless one did. However, eventually this probability reaches the same order of magnitude as the probability that the noisy corrector itself makes an error at which time the probability of error per digit reaches an equilibrium value. Notice that although such an equilibrium value is attained, it is still possible for errors to occur in sufficient digits so that a memory failure results. If a memory failure does occur, a propagation failure will also occur and the bound on the probability of error per digit will no longer be valid.

In light of this intuitive argument, we expect that the time at which the probability of error per digit is at its maximum value is just before the end of the first correcting cycle. In Appendix A upper bounds are computed on this probability evaluated just before the end of the first and successive correcting cycles. It is shown that, provided no propagation failure occurs, these bounds form a monotonically decreasing sequence; hence, the bound on this probability evaluated just before the end of the first correcting cycle, denoted by p_0, is the desired bound.

The next step is to bound the probability that the initial propagation failure occurs at some particular time. A propagation failure occurs whenever the error pattern in the memory is related to the component errors that occurred m or more iterations previously. In most cases the error pattern depends only on component errors that occurred in the last few iterations since any digit errors caused by previous component errors would have already been corrected. In order for a propagation failure to occur, the effect of component errors must have propagated from one iteration to the next for at least

m iterations. Thus we expect that the probability of such a propagation decreases as m is increased and, since increasing k increases m, that the probability of a propagation failure can be made arbitrarily small by making k sufficiently large. The explicit relationship between the probability of the initial propagation failure and k is derived in Appendix B. The result is

$$\Pr[\text{initial propagation failure}] < Ck^{-\beta+1}$$

where C and β depend only on the parameters of the code (J and K) and the bound on the probability of error per digit, p_0; hence, C and β are constants for any particular sequence of memories.

Some typical values for β are shown in Table I. For example, if $J = 14, K = 15$, and $p_0 = 10^{-8}$ then $\beta = 7.55$; therefore, in this case, the probability that the first propagation failure occurs at some particular time is upper bounded by a function that decreases as the sixth power of the information storage capability of the memory. By choosing the information storage capability sufficiently large, this probability can be made arbitrarily small.

For the moment, let us assume that neither a memory failure nor a propagation failure has occurred within M_k during the time interval $0 \leqq t < \mathcal{L}'\tau$. We now use the bound on the probability of the initial propagation failure to find an upper bound on the probability that either the initial memory failure or the initial propagation failure occurs at $t = \mathcal{L}'\tau$. To determine whether the initial memory failure occurs at $t = \mathcal{L}'\tau$ we must imagine that M_k becomes noiseless at $t = \mathcal{L}'\tau$ and that the noiseless correcting network within M_k performs m more iterations. As explained previously, the initial memory failure can occur at $t = \mathcal{L}'\tau$ only if the initial propagation failure occurs at $t = \mathcal{L}'\tau$ or during the m noiseless iterations performed after $t = \mathcal{L}'\tau$. Thus the sum of the

TABLE I—TYPICAL VALUES OF $\beta'(J, K, p_0)$ AND $\beta(J, K, p_0)$.

J	K	R (lower bound)	p_0	β	β'
4	5	0.20	10^{-8}	2.66	0.66
6	7	0.14	10^{-8}	3.91	1.91
8	9	0.11	10^{-8}	4.95	2.95
10	11	0.09	10^{-8}	5.89	3.89
12	13	0.07	10^{-8}	6.75	4.75
14	15	0.06	10^{-8}	7.55	5.55

probabilities that the initial propagation failure occurs at $t = \mathfrak{L}'\tau$, $(\mathfrak{L}' + 1)\tau, \cdots, (\mathfrak{L}' + m)\tau$ is an upper bound on the desired probability. Therefore,

$$\Pr[\text{initial memory failure or initial propagation failure at } t = \mathfrak{L}'\tau] < (m + 1) \cdot C \cdot k^{-\beta+1}.$$

Gallager[10] has shown that

$$m < \frac{\log N}{\log [(J - 1)(K - 1)]}.$$

Therefore,

$$\begin{aligned} m + 1 &< \frac{\log N}{\log [(J - 1)(K - 1)]} + 1 \\ &< \frac{\log \left[\frac{k}{1 - J/K}\right]}{\log [(J - 1)(K - 1)]} + 1 \\ &< \log \left[\frac{k}{1 - J/K}\right] \quad \text{if} \quad k > 2. \end{aligned}$$

(Note : $K > J \geqq 4$).

The probability that the initial memory failure occurs during the time interval $0 \leqq t \leqq \mathfrak{L}\tau$ is upper bounded by the sum of the probabilities that either the initial memory failure or the initial propagation failure occurs at $t = 0, \tau, 2\tau, \cdots, \mathfrak{L}\tau$. This bound equals

$$\begin{aligned} &\Pr[\text{failure during time interval } 0 \leqq t \leqq \mathfrak{L}\tau] \\ &\quad < (\mathfrak{L} + 1) \cdot C \cdot \log \left[\frac{k}{1 - J/K}\right] \cdot k^{-\beta+1} \\ &\quad < (\mathfrak{L} + 1) \cdot \frac{C}{1 - J/K} \cdot k^{-\beta+2} \\ &\quad = (\mathfrak{L} + 1) \cdot C' \cdot k^{-\beta'} \end{aligned}$$

where

$$C' \triangleq \frac{C}{1 - J/K} \quad \text{and} \quad \beta' \triangleq \beta - 2.$$

To show that there is a stable sequence of noisy memories, we must show that it is possible to choose J, K, and p_0 such that $\beta' > 0$. Recall that when we speak of a particular sequence of memories, $\{M_k\}$,

we mean that k is a variable whereas the values of J, K, and the probabilities of component errors are all fixed. Certain conditions have already been imposed on J, K and p_0. These conditions are:

(i) $\quad J = 2l, \qquad l = 2, 3, 4 \cdots$

(ii) $\quad K > J$

(iii) $\quad p_0 > 2p_r + p_c$

(iv) $\quad p_0 > \binom{J-1}{J/2}[(K-1)(p_0 + p_a)]^{J/2} + p_d + p_r$

(v) $\quad p_0 > p_a$

where p_a, p_d, p_r and p_c must all be fixed and greater than zero. To demonstrate that there are values of J, K and p_0 which satisfy these conditions and for which $\beta' > 0$, consider an example where $p_a = p_d = p_r = p_c = 10^{-9}$ and where $p_0 = 10^{-8}$. For this example conditions (iii), (iv), and (v) are satisfied for all reasonable values of J and K (that is, where $J < K \ll p_0^{-1}$). The values of $\beta'(J, K, p_0 = 10^{-8})$, which correspond to some typical values of J and K, are shown in Table I. For all the values of J and K which are considered, the value of β' is greater than zero. Therefore, in all these cases, the probability of a memory failure in M_k at $t = \mathcal{L}\tau$ can be made arbitrarily small by making k sufficiently large. This proves that there are stable sequences of noisy memories.

3.4 *Stability of Memories Constructed from Failure-Prone Components*

Thus far we have restricted our attention to memories in which component malfunctions are assumed to be statistically independent both from one component to another and from one use of a particular component to another use. These assumptions form the basis of a mathematical model for the component malfunctions that are commonly attributed to "noise" in the system. Unfortunately, the model does not adequately represent the most common type of component malfunction that one finds in computing systems: malfunctions where individual components fail permanently.

To see whether memories of the type considered in the previous section can be stable, if their components fail permanently, let us recall the proof of the stability theorem for noisy memories. In carrying out this proof, it is necessary to show that there are types of memories for which the probability of error per digit can be bounded (see Appendix A). If one attempts to find memories in which the

components fail permanently and for which a similar bound on the probability of error per digit exists, it becomes clear that memories constructed from these components cannot have such a time independent bound of value less than ½. This is because the probability that any particular component has failed increases with time given that components fail permanently; hence, if one hypothesizes any bound on the probability of error per digit which is less than ½, it is always possible to find a time at which the hypothesized bound is violated, showing that such a bound cannot exist. By using arguments such as those in Section 2.3, one can obtain a statistical model for the errors after each correcting cycle in terms of an equivalent channel whose capacity decreases with time. Since the capacity decreases, it is always possible to find a time at which the capacity of the equivalent channel is below the information rate of the code used in storing the information in the memory, thus precluding the possibility of effective error correction and hence the possibility of stability.

Fortunately, in most "nonspace" applications, regular maintenance is performed on computing systems, that is, components which have failed are periodically replaced with good ones. Numerous specific failure probability distributions and maintenance schemes could be considered individually; however, for the purpose of this analysis we consider instead a general case which includes many of the common probability distributions and maintenance schemes. This general case is the one for which it is possible to upper bound, during each correcting cycle, the probability that each component has failed up to or during that correcting cycle. For example, suppose that each component is replaced every T seconds and that p_f represents the probability that any particular component initially fails during any particular correcting cycle. For this example, the desired upper bound on the probability of component failure equals $T \cdot p_f$ which, by appropriate choice of T and p_f, can be made less than ½. Notice that we are free to choose both T and p_f since, as before, we are only trying to show that there exists some memory of the type under consideration which is stable.

One can now perform an analysis identical to that performed in the previous section. Since the technique used to prove the stability theorem for noisy memories does not rely upon the assumption that component errors are statistically independent from use to use, precisely the same technique used previously can be used here to prove that periodically maintained memories can be stable. In fact, in most

cases, the changes required to make this proof apply when components fail permanently merely involve replacing the words "component error" with the words "component failure."

One change which requires some reinterpretation of terms involves the concept of a propagation failure. This concept was introduced for the purpose of establishing a condition under which the parity checks used to estimate any particular digit would be conditionally independent. To facilitate an intuitive discussion of propagation failures, the original definition of a propagation failure was made more general than necessary for the mathematical analysis. This analysis, in Appendix B, uses the fact that undesirable statistical dependencies can occur only if the effects of previous component malfunctions form a Δ propagation path in some parity check set tree. It is now desirable to redefine a propagation failure in terms of the formation of such a Δ propagation path. This is because permanent component failures can result in recurrent errors in a particular digit during m or more iterations thus causing a propagation failure, according to the original definition. However, since these recurrent errors do not lead to the undesirable statistical dependencies unless they also correspond to an undesirable Δ propagation path, the original definition of a propagation failure should be changed to exclude recurrent errors.

Once these changes have been made, if one represents the bounds on the probabilities of component failures by the same symbols that were used previously to represent the actual probabilities of component errors during each iteration, not only is the method of proving the stability theorem identical to that used previously but so are the forms of all the results. Thus one proves the following theorem.

Theorem 2: There is a stable sequence of memories where every component in each of the memories in the sequence has a nonzero probability of permanent failure but where components which have failed are periodically replaced with good ones.

IV. CONCLUSIONS

In Section I we compare the results obtained by Shannon concerning the reliability of communication systems with the results obtained by von Neumann, Elias, Winograd, and Cowan concerning the reliability of computing systems. Shannon's results were basically different from the other results considered. Shannon was able to show that it is possible to design arbitrarily reliable communication systems through which information can be transmitted at a nonzero

information rate. The maximum rate for which the probability of error can be made arbitrarily small is called the capacity of the communication channel. In analogy with this result, one might expect that it should be possible to design arbitrarily reliable computing systems which have a bounded redundancy. Unfortunately, none of the computing systems proposed previously has this property; therefore, none of these computing systems has a nonzero "computing capacity."

In Sections II and III we restrict our attention to one part of a computing system, namely the memory. It is shown that there are noisy memories and periodically repaired memories constructed from failure-prone components which have the property that the probability of failure can be made arbitrarily small for certain bounded values of the redundancy. The memories which have this property are called "stable memories." This result is analogous to Shannon's result and is basically different from the other results obtained thus far concerning the reliability of computing systems. The fact that it is possible to make a memory arbitrarily reliable while keeping its redundancy bounded indicates that a memory has an "information storage capacity" analogous to the capacity of a communication channel. The information storage capacity of a particular memory equals the reciprocal of the minimum redundancy for which the memory is stable; hence it can be expressed in bits per component. It is a function of the probabilities of error for the components within the memory. The method used to prove the stability theorem does not allow one to compute an explicit value for the information storage capacity of the memories which were considered; however, the fact that these memories can be stable indicates that they do have a nonzero information storage capacity.

V. ACKNOWLEDGMENTS

I would like to sincerely thank Professor Robert M. Fano, who supervised this work, for suggesting the approach to the problem and supplying guidance and encouragement throughout the research. I also wish to thank Professor Robert G. Gallager who was extremely helpful in connection with this work. Many of the results presented here are based on results originally obtained by Professor Gallager. Finally, I wish to thank Professors Peter Elias and Claude E. Shannon who helped to formulate the problem and contributed valuable suggestions and constructive criticism.

APPENDIX A

A Bound on the Probability of Error Per Digit

The first step in computing a bound on the probability of a propagation failure is to bound the probability of error for any digit stored in the registers within the memory. We assume that initially one set of J code words is transmitted across the noisy channels and inserted into these noisy registers. Some time during the next τ seconds the correcting network reads the contents of the registers and starts to perform the first correcting cycle on the newly inserted digits. The time at which this first correcting cycle starts is denoted by $t = 0$ and successive correcting cycles start at $t = \tau, 2\tau, 3\tau, \ldots$. We denote the instant just before the end of the first correcting cycle by $t = \tau - \delta$. If a digit is in error at $t = \tau - \delta$, at least one of the following events must have occurred:

(*i*) An error was made in transmitting the digit across the BSC. The probability of this event is p_c.

(*ii*) The digit was changed because of a component error in the register which occurred either during the time interval $-\tau < t \leqq 0$ or $0 < t \leqq \tau - \delta$. The probability of this event is less than $2p_r$.

Therefore, by the union bound, the probability of error per digit at $t = \tau - \delta$ is bounded by

$$\Pr[\text{digit} = \epsilon \text{ at } t = \tau - \delta] < 2p_r + p_c < p_0$$

where the parameter p_0 has been introduced to simplify the form of the results. Other conditions on p_0 will be imposed later.

Next let us compute a bound on the probability of error per digit at $t = 2\tau - \delta, 3\tau - \delta, \ldots$. If the digit d_0 is in error at any one of these times, at least one of the following events must have occurred:

(*i*) A set of $J/2$ parity checks used to estimate d_0 were in error during the last correcting cycle performed on d_0.

(*ii*) The decision device made an error during the last correcting cycle.

(*iii*) An error occurred while d_0 was stored in the register.

There are $\binom{J-1}{J/2}$ possible events of the first type. We now compute the probability that any one of these events occurred. If the ith parity check used to estimate d_0, c_i, were in error there must have been at least one error among the $K - 1$ adders used to evaluate this parity check, or at least one error among the $K - 1$ digits denoted by d_{i1},

d_{i2} , $\cdots$, d_{iK-1} . According to the union bound, the probability that c_i (for any $0 < i \leqq J - 1$) is in error is bounded by

$$\Pr [c_i = \epsilon] < \sum_{j=1}^{K-1} \Pr [d_{ij} = \epsilon] + (K - 1)p_a .$$

During the first correcting cycle $\Pr [d_{ij} = \epsilon] < p_0$ for all $0 < j \leqq K - 1$; therefore, for this iteration

$$\Pr [c_i = \epsilon \text{ during first correcting cycle}] < (K - 1)p_0 + (K - 1)p_a .$$

To compute the probability of error per digit just before the end of the second iteration, we use the fact that, during the first m iterations, the structure of the code guarantees that errors in the parity checks used to estimate any particular digit are statistically independent. To see this, consider the parity-check set tree rising from the digit d_0 as shown in Fig. 6. Each node on this tree represents a particular digit. With each digit there is associated a set of components used to compute each estimate of the digit. Just as the tree represents a history of the digits which have been involved in the computation of the successive estimates of d_0 , it also represents a history of the components which have been involved in the computation of these estimates. The later interpretation is more useful for our purposes since it is the components which cause the errors. If the code has m independent iterations, all the digits on the first m tiers of the tree must be different and all the components associated with these digits must be different also. There are $(K - 1)(J - 1)$ digits on the first tier of this tree. Provided that $m \geqq 1$, the errors in these digits after the first iteration must be statistically independent since the digits are all different, and hence the components used to compute the estimates of these digits are all different. (It is assumed that errors in different components are statistically independent.) In general, the errors in the digits on the first tier of the tree, and hence the $J - 1$ parity checks, are statistically independent provided that the sets of components used to compute the estimates of these digits are disjoint. The structure of the tree guarantees that this condition will be satisfied for the first m iterations.

Since, during the first m iterations, the errors in the parity checks used to estimate any particular digit are statistically independent, the probability that a set of $J/2$ parity checks is in error equals the product of the probabilities that each one of these $J/2$ parity checks is in error. Thus the probability of error per digit at $t = 2\tau - \delta$ is

bounded by

$$\Pr[\text{digit} = \epsilon \quad \text{at} \quad t = 2\tau - \delta] < \binom{J-1}{J/2}[(K-1)(p_0 + p_a)]^{J/2} + p_d + p_r \triangleq p_1 .$$

Since we are not attempting to show that all memories of the type under consideration are stable but merely that there exist some memories of this type which are stable, we shall restrict our interest to those memories for which it is possible to make $p_1 < p_0$. This is not a serious restriction since in most cases there is no difficulty in bounding p_1 by p_0. For example, if

p_a, p_r and $p_d = 10^{-9}$, $p_0 = 10^{-8}$, $J = 14$ and $K = 15$, then $p_1 \approx 2.10^{-9}$ illustrating one case where $p_1 < p_0$.

Precisely the same argument can be used to obtain a bound on the probability of error per digit at $t = 3\tau-\delta, 4\tau-\delta, \ldots, (m+1)\tau-\delta$. The results are:

$$\Pr[\text{digit} = \epsilon \quad \text{at} \quad t = 3\tau - \delta] < \binom{J-1}{J/2}[(K-1)(p_1 + p_a)]^{J/2} + p_d + p_r \triangleq p_2 < p_1 < p_0 .$$

Similarly

$$\Pr[\text{pigit} = \epsilon \quad \text{at} \quad t = (m+1)\tau - \delta] < \binom{J-1}{J/2}[(K-1)(p_{m-1} + p_a)]^{J/2} + p_d + p_r \triangleq p_m < p_{m-1} < \cdots < p_1 < p_0 .$$

If no propagation failure occurs at $t = m\tau$, the error pattern evaluated at that time depends on the component errors that occurred during the previous m iterations, but not on the original errors that were present at $t = 0$. Imposing this condition changes the probability of error per digit at $t = (m+1)\tau-\delta$; however, as we now show, the probability computed above is an upper bound on this conditional probability. Using Bayes rule,

$$\Pr[\text{digit} = \epsilon \mid \text{no propagation failure}] = \frac{\Pr[\text{no propagation failure} \mid \text{digit} = \epsilon] \cdot \Pr[\text{digit} = \epsilon]}{\Pr[\text{no propagation failure}]}.$$

It is easily shown, using techniques similar to those in Appendix B, that

$$\Pr[\text{no propagation failure} \mid \text{digit} = \epsilon] \leqq \Pr[\text{no propagation failure}],$$

therefore

$$\Pr[\text{digit} = \epsilon \mid \text{no propagation failure}] \leqq \Pr[\text{digit} = \epsilon].$$

If no propagation failure has occurred, the errors in the set of parity checks used to estimate any particular digit are conditionally independent. This is because, if there is no propagation failure, the errors in each set of parity checks depend on errors in unrelated, disjoint sets of components. Thus the technique used to bound the probability of error per digit during the first m iterations can be applied after the mth iteration provided that no propagation failure has occurred. The general result is

$$\begin{aligned}\Pr[\text{digit} = \epsilon \text{ at } t = (i+1)\tau - \delta \mid \text{no propagation failure}] &< \binom{J-1}{J/2}[(K-1)(p_{i-1}+p_a)]^{J/2} + p_d + p_r \\ &\triangleq p_i < p_{i-1} < \cdots < p_i < p_0 .\end{aligned}$$

This monotonically decreasing sequence of bounds shows that the probability of error per digit is upper bounded by p_0 provided that it is possible to choose p_0, J, and K such that $p_1 < p_0$ and provided that no propagation failure occurs. In general, the probability of error per digit is upper bounded by p_i provided that the memory has performed i or more correcting cycles and provided that the conditions stated above are satisfied.

APPENDIX B

The Probability of a Propagation Failure

Intuitively, the concept of a propagation failure is very simple. A propagation failure occurs whenever the present error pattern in the registers is in some way related to the component errors that occurred more than m iterations before, where m is the number of in-

dependent iterations. This intuitive concept can be made more precise by defining "error configurations" which are hypothetical error patterns that are functions of some subset of the actual set of component errors. In particular, a *type i error configuration*, for any $i > 0$, is a $J \cdot N$-tuple corresponding to the error pattern that would be present if there had been no component errors before the last i iterations. If no propagation failure has occurred, all error configurations of type m and higher must be identical, thus we are really interested in the difference between error configurations. For this reason, it is more convenient to restate the definition of a propagation failure in terms of "Δ configurations" where, for all $i > 0$, the *type i Δ configuration* is the difference between the type i and type $i+1$ error configurations. The type 0 Δ configuration is defined to be equal to the type 1 error configuration. A propagation failure occurs if there are one or more 1's in any Δ configuration of type m or higher.

Let us consider the situations which, for any i, lead to a 1 in the type i Δ configuration evaluated at some particular time, say $t = L\tau$. Suppose that there is a 1 in the position corresponding to the digit d_0 in this Δ configuration. This means that the value of the digit d_0 in the type i error configuration is different from that in the type $i+1$ error configuration where both configurations are evaluated at $t = L\tau$. This change in the value of d_0 must be related to component errors that occurred during the time interval $(L-i-1)\tau < t < (L-i)\tau$ since all the other component errors upon which the type i and type $i+1$ error configurations are based are the same. We refer to the component errors that occurred during the time interval $(L-i-1)\tau < t < (L-i)\tau$ as the *controlling errors* for the type i Δ configuration evaluated at $t = L\tau$ and we say that the value of d_0 at $t = L\tau$ has been changed by these controlling errors.

If the value of d_0 is changed at $t = L\tau$, the controlling errors must have changed at least one of the digits used to estimate d_0. These are the digits on the first tier of the parity-check set tree rising from d_0, and changes in them are represented by 1's in the appropriate positions in the type $i-1$ Δ configuration evaluated at $t = (L-1)\tau$. In general these controlling errors must have caused changes in some digits on the lth tier of the parity-check set tree, the changes being represented by 1's in the appropriate positions in the type $i-l$ Δ configuration evaluated at $t = (L-l)\tau$. These changed digits define at least one continuous path in the parity-check set tree rising i tiers from d_0. We call these paths Δ *propagation paths*. The particular Δ propagation path which has just been described is referred to as the i-tier

Δ propagation path rising from d_0 at $t = L\tau$ (see Fig. 8). Every 1 in a type i Δ configuration must have at least one i-tier Δ propagation path associated with it. To bound the probability that there is a 1 in the entry corresponding to the digit d_0 in the type i Δ configuration evaluated at $t = L\tau$, we shall bound the probability that one or more i-tier Δ propagation paths rise from the digit d_0 at $t = L\tau$.

Since Δ propagation paths must be continuous, for each 1 in a type i Δ configuration evaluated at $t = L\tau$ there must have been at least one 1 in a type $i-1$ Δ configuration evaluated at $t = (L-1)\tau$. If no propagation failure occurred before $t = L\tau$, the Δ configurations of type m and higher must have beeen all zero for all $t < L\tau$. This implies that the Δ configurations of type $m+1$ and higher must be all zero at $t = L\tau$. Hence, the only way that the initial propagation failure can occur at $t = L\tau$ is if there is a 1 in the type m Δ configuration evaluated at that time. Therefore, to bound the probability that the first propagation failure occurs at some particular time, we need

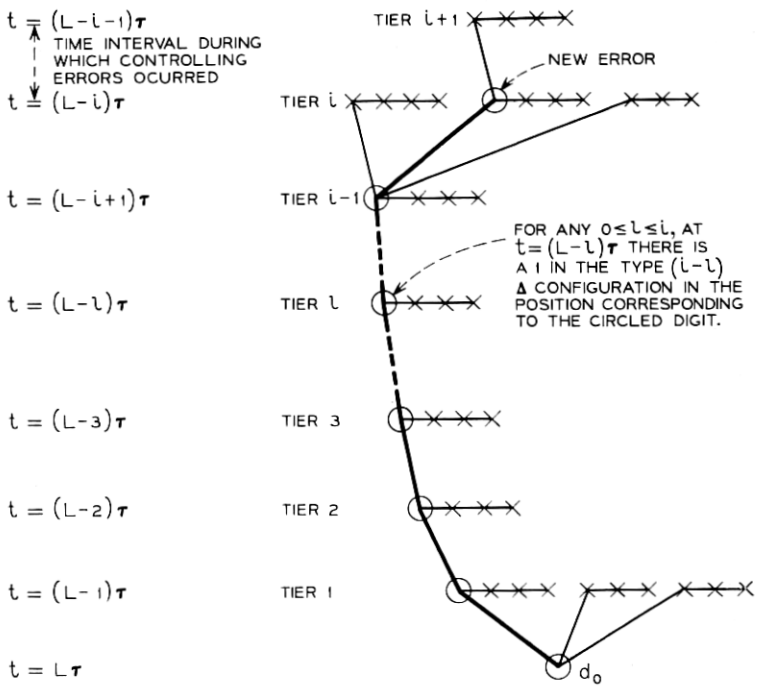


Fig. 18 — An example of an i-tier Δ propagation path rising from d_0 at $t = L\tau$.

only bound the probability of one or more 1's in the type m Δ configuration evaluated at that time. Since we assume that the memory fails whenever the first propagation failure occurs, we shall never be concerned with computing the probability of a 1 in any Δ configuration of type $m+1$ or higher. It is important that we can restrict our attention to the type m Δ configuration since this Δ configuration can be computed for any particular time by considering only the component errors that occurred during the previous $m+1$ correcting cycles. The first of these correcting cycle results in statistically independent digit errors and, as explained in Appendix A, the structure of the code guarantees that during the next m correcting cycles the errors in the parity checks used to estimate any digit are statistically independent.

To bound the probability that the initial propagation failure occurs at $t = L\tau$, we must compute a bound on the probability of one or more 1's in the type m Δ configuration evaluated at $t = L\tau$. This is done by bounding the probability that an m-tier Δ propagation path rises from one or more of the $J \cdot N$ digits in the registers within the memory at $t = L\tau$. At first we restrict our attention to one particular digit d_0. A bound is computed on the probability that an m-tier Δ propagation path rises from d_0 at $t = L\tau$. The first step in this computation is to bound the probability that the component errors that occurred during the time interval $(L-m-1)\tau < t < (L-m)\tau$ would cause any particular digit on the mth tier of the parity-check set tree rising from d_0 to be in error at $t = (L-m)\tau$. Any m-tier Δ propagation path rising from d_0 at $t = L\tau$ must terminate on one of these errors which we call *new errors*. The next step is to bound the probability that at $t = (L-m+i)\tau$ an i-tier Δ propagation path rises from any particular digit on the $(m-i)$th tier of the parity-check set tree rising from d_0. This probability is denoted by $Pr[d_{m-i}=\Delta_i]$. By substituting m for i, we obtain a bound on the probability that at $t = L\tau$ an m-tier Δ propagation path rises from the digit d_0 (that is, $Pr[d_0 = \Delta_m]$). The probability that an m-tier Δ propagation path rises from one or more of the $J \cdot N$ digits at $t = L\tau$ is upper bounded by $J \cdot N \cdot Pr[d_0 = \Delta_m]$.

The first step, namely bounding the probability of a new error at $t = (L-m)\tau$, is particularly simply. In Appendix A we computed a bound on the probability of error per digit which was denoted by p_0. Since this bound was computed by considering all possible errors that could exist at a particular time, it must certainly be a bound on the

probability of a new error at some particular time. Therefore, p_0 is a bound on the probability that a new error occurs at $t = (L-m)\tau$.

The next step is to bound Pr $[d_{m-i} = \Delta_i]$ for all $1 \leqq i \leqq m$. Let us consider a particular digit on the $(m - i)$th tier of the parity-check set tree rising from d_0 and denote this digit by d_{m-i}. Now consider the conditions which must be satisfied if the value of d_{m-i} is changed by the controlling errors; that is, if an i-tier Δ propagation path rises from d_{m-i} at $t = (L - m + i)\tau$.

To describe this "change" in more detail, we must consider two sets of component errors. One is the set of component errors that occurred since $t = (L - m - 1)\tau$ and the other is the set of component errors that occurred since $t = (L - m)\tau$ [assume that no component errors occurred before $t = (L - m - 1)\tau$ and $t = (L - m)\tau$, respectively]. When we say that the value of d_{m-i} has been changed by the controlling errors, we mean that the value of d_{m-i} at $t = (L - m + i)\tau$ is correct when it is computed under the assumption that one of these sets of component errors actually occurred, whereas it is incorrect when it is computed under the assumption that the other set of errors actually occurred. There are two necessary conditions for this change. Assume that the value of d_{m-i} was changed by the controlling errors. Denote the set of component errors for which $d_{m-i} = \epsilon$ by $\mathcal{S}_\epsilon$ and the set for which $d_{m-i} \neq \epsilon$ by $\mathcal{S}_{\text{correct}}$. If the value of d_{m-i} is changed by the controlling errors, both of the following conditions must be satisfied:

(*i*) There must have been at least one parity check used to estimate d_{m-i} which was wrong [at $t = (L - m + i)\tau$] under the assumption that $\mathcal{S}_\epsilon$ occurred but which was correct under the assumption that $\mathcal{S}_{\text{correct}}$ occurred. Denote one of these parity checks by c_Δ.

(*ii*) On the basis of the errors in the set $\mathcal{S}_\epsilon$, $J/2 - 1$ or more parity checks other than c_Δ must have been wrong at $t = (L - m + i)\tau$.

In order for condition i to be satisfied, the value of at least one of the $(J-1)(K-1)$ digits immediately above d_{m-i} in the parity-check set tree must have been changed [at $t = (L-m+i-1)\tau$] by the controlling errors. The probability of such a change has been denoted by $Pr[d_{m-i+1} = \Delta_{i-1}]$. Therefore, the probability that the value of one or more of these digits was changed is upper bounded by

Pr [condition i is satisfied]

$<$ Pr [value of any digit immediately above d_{mi} is changed by controlling errors]

$<$ $(J - 1)(K - 1)$ Pr $[d_{m-i+1} = \Delta_{i-1}]$.

A bound on the probability that condition *ii* is satisfied was derived in Appendix A. This bound equals

$$\Pr[\text{condition } (ii) \text{ is satisfied}] < \binom{J-2}{J/2-1}[(K-1)(p_0+p_a)]^{J/2-1}.$$

As explained previously, the structure of the code guarantees that parity-check errors are independent; hence, during the m iterations of interest, these two conditions are independent. Therefore, the probability that both conditions are satisfied, which is a bound on $Pr[d_{m-i} = \Delta_i]$, is given by

$$\Pr[d_{m-i} = \Delta_i] < (J-1)(K-1)\Pr[d_{m-i+1} = \Delta_{i-1}] \cdot \binom{J-2}{J/2-1}[(K-1)(p_0+p_a)]^{J/2-1}.$$

Substituting $i = 1, 2 \ldots m$, we obtain

$$\Pr[d_{m-1} = \Delta_1] < (J-1)(K-1)p_0\binom{J-2}{J/2-1}[(K-1)(p_0+p_a)]^{J/2-1}$$

$$\Pr[d_{m-2} = \Delta_2] < (J-1)(K-1)\Pr[d_{m-1} = \Delta_1] \cdot \binom{J-2}{J/2-1}[(K-1)(p_0+p_a)]^{J/2-1}$$

$$< p_0\left\{(J-1)(K-1) \cdot \binom{J-2}{J/2-1}[(K-1)(p_0+p_a)]^{J/2-1}\right\}^2$$

$$\vdots$$

$$\Pr[d_0 = \Delta_m] < p_0\left\{(J-1)(K-1) \cdot \binom{J-2}{J/2-1}[(K-1)(p_0+p_a)]^{J/2-1}\right\}^m.$$

Gallager has found a technique for constructing low-density parity-check codes[10] with m, the number of independent iterations, bounded by

$$\frac{\log\left[\frac{N}{2K} - \frac{N}{2J(K-1)}\right]}{2\log[(J-1)(K-1)]} \leqq m \leqq \frac{\log N}{\log[(J-1)(K-1)]}.$$

Substituting this lower bound into the equation for $Pr[d_0 = \Delta_m]$ gives

$$\Pr[d_0 = \Delta_m] < p_0\left\{(J-1)(K-1)\binom{J-2}{J/2-1}\cdot[(K-1)(2p_0)]^{J/2-1}\right\}^{\log[(N/2K)-[N/2J(K-1)]]/2\log[(J-1)(K-1)]}$$

$$= p_0\left\{\left[\frac{1}{2K} - \frac{1}{2J(K-1)}\right]N\right\}^{-\beta}$$

where

$$\beta \triangleq -\frac{\log\left\{(J-1)(K-1)\binom{J-2}{J/2-1}[(K-1)(2p_0)]^{J/2-1}\right\}}{2\log(J-1)(K-1)}.$$

We have assumed that p_0 has been chosen such that $p_0 > p_a$. For any L, probability that the initial propagation failure occurs at $t = L\tau$ is bounded by

$$\Pr[\text{initial propagation failure occurs at } t = L\tau]$$

$$= 0 \qquad \text{for } L < m$$

$$< J \cdot N \cdot \Pr[d_0 = \Delta_m] \qquad \text{for } L \geqq m$$

since, by definition, a propagation failure cannot occur before $t = m\tau$.

We have chosen to number the memories according to their information storage capability, k. We can express this result in terms of k by noting that $N = k/R$ and $1 - J/K \leqq R \leqq 1$; therefore,

$$\Pr[\text{initial propagation failure occurs at } t = L\tau]$$

$$= 0 \qquad \text{for } L < m$$

$$< J\cdot\left[\frac{k}{1-\frac{J}{K}}\right]\cdot p_0\cdot\left\{\left[\frac{1}{2K} - \frac{1}{2J(K-1)}\right]k\right\}^{-\beta} \qquad \text{for } L \geqq m$$

$$= C \cdot k^{-\beta+1}$$

where

$$C \triangleq \frac{J}{1-J/K}\cdot p_0\cdot\left[\frac{1}{2K} - \frac{1}{2J(K-1)}\right]^{-\beta}.$$

Both C and β are functions of J, K and p_0. For any particular sequence of memories, J, K and p_0 will all be constants. For example, if

$J = 14$, $K = 15$, and $p_0 = 10^{-8}$, then $\beta = 7.55$; therefore, in this case, the probability that the first propagation failure occurs at $t = L\tau$ (for any L) is bounded by a function that decreases as the sixth power of the information storage capability of the memory. By choosing the information storage capability sufficiently large, this probability can be made arbitrarily small.

REFERENCES

1. von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," in *Automata Studies*, ed. C. E. Shannon and J. McCarthy, Princeton, New Jersey: Princeton University Press, 1956, pp. 43–98.
2. Allanson, J. T., "The Reliability of Neurons" Proc. First Congress on Cybernetics, 1956, Paris: Gauthier-Villars, 1959, pp. 687–694.
3. Shannon, C. E., "A Mathematical Theory of Communication," B.S.T.J., *27*, Nos. 3 and 4 (July and October 1948), pp. 379–423, and 623–656.
4. Elias, P., "Computation in the Presence of Noise," IBM J. Research and Development, *2* (October 1958), pp. 346–353.
5. Elias, P., "Coding for Two Noisy Channels" in *Information Theory*, ed. Colin Cherry, New York: Academic Press, 1956, pp. 61–74.
6. Wozencraft, J. M., and Reiffen, B., *Sequential Decoding*, New York: Technology Press and John Wiley and Sons, Inc., 1961.
7. Fano, R. M., "A Heuristic Discussion of Probabilistic Decoding," IEEE Trans. Inform. Theory, *IT-9* (April 1963), pp. 64–74.
8. Winograd, S. and Cowan, J. C., *Reliable Computation in the Presence of Noise*, Cambridge, Massachusetts: MIT Press, 1963.
9. Gallager, R. G., "A Simple Derivation of the Coding Theorem and Some Applications," IEEE Trans. Inform. Theory, *IT-11* (January 1965), pp. 3–18.
10. Gallager, R. G., *Low-Density Parity-Check Codes*, Cambridge, Massachusetts: MIT Press, 1963.