

# Node Synchronization for the Viterbi Decoder

G. Lorden<sup>1</sup>, R. McEliece<sup>2</sup>, and L. Swanson  
Communications Systems Research Section

*At very low signal-to-noise ratios such as those that Voyager 2 will encounter at Uranus, the performance of the Reed-Solomon/Viterbi concatenated coding system could be seriously degraded by loss of node synchronization by the Viterbi decoder. In this article we will describe this problem and show that it can be almost completely avoided with a simple outboard hardware "node synchronizer." This device makes statistical decisions about node sync based on the hard-quantized undecoded data stream. We show that in a worst-case Voyager-like environment, our method will detect and correct a true loss of node sync (thought to be a very rare event) within several hundred bits; many of these outages will be corrected by the RS code. At the same time, the mean time between false alarms for our technique is on the order of several years.*

## I. Introduction

This paper deals with the problem of detecting and correcting losses of node synchronization in convolutionally encoded data. We are motivated by our desire to restore the loss of telemetry on future NASA deep-space missions which has been predicted in Refs. 1 and 2, and has been seen in hardware tests (Ref. 3). For definiteness, this report will only deal with the problem as it occurs on the present Voyager 2 mission, but our results will apply equally to any mission in which the telemetry is protected by a Reed-Solomon/Viterbi concatenated coding system.

On Voyager, the high-rate downlink telemetry is protected by a  $K = 7$ , rate  $1/2$  convolutional code concatenated with a depth-4 interleaved (255,223) Reed-Solomon code. In principle this combination provides excellent error protection (bit error probability  $1.0E-6$ ) for Voyager's highly sensitive imag-

ing data, at bit signal-to-noise ratios as low as 2.9 dB. Since the rate of the outer code is  $223/255 = -0.6$  dB, when the overall SNR is 2.9 dB, the inner convolutional code is operating at about 2.3 dB.

However, in practice, the performance of the concatenated system is significantly worse than theoretical predictions. One problem is carrier-loop jitter, which degrades performance by 0.5 dB or more (Ref. 1). This means that if the system bit SNR remains at the nominal 2.9-dB value, the inner convolutional code must operate at less than 2.0 dB. This is a value much lower than that for which the DSN's hardware Viterbi decoders were designed. In this demanding environment, the Viterbi decoder's internal node synchronization hardware, whose function is to detect and correct true external losses of node sync, is prone to produce false alarms i.e., spurious losses of node sync, and send useless data to the Reed-Solomon decoder until node sync is reestablished. In Ref. 2 it was shown that this hardware problem can degrade Voyager's performance by a further 1.0 dB or more.

<sup>1</sup>Consultant, Caltech Mathematics Department

<sup>2</sup>Consultant, Caltech Electrical Engineering Department

In this article we shall show that this data loss due to spurious node sync loss in the Viterbi decoder is completely avoidable. Our proposed solution involves disabling the Viterbi decoder's internal synchronization hardware and implementing an external node sync algorithm. Our algorithm is easy to implement and depends on likelihood calculations based on observations of the hard-quantized encoded data stream. In a worst-case Voyager environment, our method will detect and correct a true loss of node sync within several hundred bits; many of these outages will be corrected by the Reed-Solomon code. On the other hand, the mean time between false alarms for our technique (which is independent of the SNR) is on the order of several years. Thus for practical purposes our technique introduces no false alarms, and the system SNR loss due to node sync problems will be eliminated, with no loss of protection against true node sync losses. As an outboard hardware device, our algorithm could be implemented on a single DSN standard single-board computer such as the iSBC 86/12, at least at data rates up to 20 kbps.

The paper is divided into three sections. In Section II, we present a functional description of our algorithm, together with a summary of the relevant mathematics. In Section III, we present some numerical performance results for our technique. They will quantify the assertions made above (mean time between false alarms, probability of uncorrectable errors due to true loss of sync, etc.). We also include in the appendices some background information.

## II. The Up-Down Counter

We adopt the following model, which has been found to be very accurate for coherent deep-space communication (Ref. 4). The information to be transmitted via the convolutional code (which in Voyager is already encoded) is a sequence  $\dots M_{-1}, M_{-2}, M_0, M_2, M_4, \dots$  of independent identically distributed random variables, each equally likely to be 0 or 1. The encoded stream  $\dots, C_{-3}, C_{-2}, C_{-1}, C_0, C_1, \dots$  is defined by the encoding equations<sup>3</sup>

$$C_{2k} = M_{2k} + M_{2k-2} + M_{2k-4} + M_{2k-6} + M_{2k-12} \pmod{2} \quad (1)$$

$$C_{2k-1} = M_{2k} + M_{2k-4} + M_{2k-6} + M_{2k-10} + M_{2k-12} \pmod{2} \quad (2)$$

<sup>3</sup>We shall illustrate all of our results for the NASA standard  $K = 7$ , rate 1/2 convolutional code, but everything generalizes easily to any rate 1/2 convolutional code.

We also define the  $\pm$  versions of the encoded stream:

$$D_j = \begin{cases} +1 & \text{if } C_j = 0 \\ -1 & \text{if } C_j = 1. \end{cases}$$

The encoded bits  $\{D_k\}$  are used to modulate a radio frequency signal, which is transmitted by Voyager to Earth. After detection and demodulation, a sequence  $\{\hat{D}_k\}$  is received, where  $\hat{D}_k = D_k + Z_k$ . The sequence  $\{Z_k\}$  is the *error* sequence. If the noise process is additive white Gaussian noise, then the sequence  $\{Z_k\}$  is i.i.d., the common distribution being Gaussian, mean zero and variance  $\sigma^2 = 1/\mu$ , where  $\mu$  is twice the symbol SNR.

The Viterbi decoder attempts to recover the message bits from the noisy code sequence  $\{\hat{D}_k\}$ . It does this by making a very efficient maximum likelihood estimate of each of the message bits  $\{M_{2j}\}$ . However, in order to operate, the Viterbi decoder *must have node sync*, i.e., it must know which of the received symbols have *even* subscripts and which have *odd* subscripts. Of course there are only two possibilities, but if the wrong hypothesis is made, the output of the Viterbi decoder will bear no useful relationship to the message stream  $\{M_{2j}\}$ .

Our algorithm will provide node sync information for the Viterbi decoder. It is based on the *hard-quantized* received sequence  $\{R_k\}$ , where

$$R_k = \begin{cases} 0 & \text{if } \hat{D}_k \text{ is } \geq 0 \\ 1 & \text{if } \hat{D}_k \text{ is } < 0. \end{cases}$$

Clearly  $R_k = C_k + E_k \pmod{2}$ , where  $\{E_k\}$  is i.i.d. and  $E_k = 0$  or 1. The *error probability*  $P_e = \Pr \{E_r = 1\}$  is given by

$$P_e = \frac{1}{\sqrt{2\pi}} \int_{\mu}^{\infty} e^{-t^2/2} dt, \quad (3)$$

where as before  $\mu = 2E_s/N_0$ .

Associated with the hard-quantized received sequence  $\{R_k\}$  is the *syndrome*, or *parity-check sequence*  $\{X_k\}$  (Ref. 5):

$$X_k = R_k + R_{k-1} + R_{k-3} + R_{k-4} + R_{k-5} + R_{k-6} + R_{k-7} \\ + R_{k-10} + R_{k-12} + R_{k-13}. \quad (4)$$

If there are no errors, i.e., if  $E_k = 0$  for all  $k$ , then it follows from (1) and (2) that  $X_k = 0$  for all even subscripts  $k$ . In fact we have explicitly

$$X_k = E_k + E_{k-1} + E_{k-3} + E_{k-4} + E_{k-5} + E_{k-6} + E_{k-7} \\ + E_{k-10} + E_{k-12} + E_{k-13} \quad k \text{ even.} \quad (5)$$

For odd  $k$ , (1) and (2) show that the  $X_k$ 's are independent and take the values 0 and 1 with probability 1/2 each, regardless of the  $\{E_k\}$  sequence. (This is shown in detail in Appendix A.)

For even  $k$ , (5) shows that  $E_k$  will be zero if and only if an even number of  $\{E_k, E_{k-1}, E_{k-3}, E_{k-4}, E_{k-5}, E_{k-6}, E_{k-7}, E_{k-10}, E_{k-12}, E_{k-13}\}$  are one. The probability of this is easily seen to be

$$\pi = \frac{1 + (1 - 2p_e)^{10}}{2}. \quad (6)$$

The foregoing describes the distribution of the received sequence and of the parity-check sequence in case node synchronization is maintained. This will be called the *in-sync hypothesis*. The *out-of-sync hypothesis* describes the situation when node synchronization is in error. In this case the  $R_i$ 's and  $X_i$ 's behave as though the subscripts were shifted by one. Thus, under the out-of-sync hypothesis it is the odd parity checks that are correct with probability  $\pi$  and the even ones that are purely random.

We assume that node synchronization has been acquired and maintained and that the in-sync hypothesis is initially true. We wish to monitor the received sequence so as to detect loss of sync, i.e., a sudden change making the received sequence obey the out-of-sync hypothesis. A method for doing this simply can be based on a general statistical technique (Ref. 6) for detecting a change in distribution. To apply this technique it is necessary to simplify the model by assuming that the parity checks  $X_i$  are independent. They are not independent for even  $i$ , but the dependence between widely separated  $X_i$ 's is slight, so calculations based on this assumption should be illustrative.

The method for detecting loss of sync is based on a *counter* with increments

$$L(X_n) = \log \frac{q(X_n)}{p(X_n)},$$

where  $p(X_n)$  and  $q(X_n)$  are the likelihoods of  $X_n$  under the in-sync and out-of-sync hypotheses, respectively.

As shown above

$$p(X_n) = \begin{cases} 1/2 & \text{if } n \text{ is odd} \\ \pi & \text{if } n \text{ is even and } X_n = 0 \\ 1 - \pi & \text{if } n \text{ is even and } X_n = 1 \end{cases}$$

whereas  $q(X_n)$  reverses the odd and even cases. Thus,

$$L(X_n) = \begin{cases} (-1)^{n+1} \log(2\pi) & \text{if } X_n = 0 \\ (-1)^{n+1} \log(2(1-\pi)) & \text{if } X_n = 1. \end{cases}$$

The counter is defined by

$$T_0 = 0$$

$$T_n = \max([T_{n-1} + L(X_n)], 0), \quad n \geq 1.$$

A threshold  $\gamma > 0$  is chosen, and the process stops the first time  $T_n \geq \gamma$ . Since at this point there is a substantial likelihood ratio in favor of the out-of-sync hypothesis, the inference is made that loss of node synchronization has occurred. This loss of node sync can be remedied by either adding or deleting a channel symbol; the node synchronizer will alternate adding and deleting symbols in order to avoid ruining frame boundaries.

The performance of such a counter for a particular  $\gamma$  is characterized by two *average run lengths* (ARL's).

- (1) The *short* ARL. This is the average number of pairs of symbols needed to reach the threshold if the out-of-sync hypothesis is true from the beginning, and
- (2) The *long* ARL. This is the average number of pairs of symbols needed to reach the threshold if the in-sync hypothesis remains true.

The *short* ARL gives an upper bound on the average time between loss of sync and its detection, since whenever loss of sync occurs the counter has a nonnegative value. The *long* ARL (or its reciprocal) describes the frequency of false detection of loss of node synchronization.

An exact determination of the ARL's will be made in Section III, for a very slight modification of the scheme described. It is instructive, however, particularly for later comparisons, to consider their asymptotic behavior as  $\gamma \rightarrow \infty$ . It turns out that

$$\text{short ARL} \sim \frac{\gamma}{I}$$

$$\gamma = m \log \frac{\pi}{1-\pi}$$

and

$$\text{long ARL} \sim C e^\gamma,$$

so that

$$\text{short ARL} \sim \frac{\log(\text{long ARL})}{I}. \quad (7)$$

Here the constant  $I$ , which is much more critical than  $C$ , is the Kullback-Leibler *information number* per pair of symbols. This number is simply the average increment of the counter per pair of symbols when the out-of-sync hypothesis is true. Since each pair of symbols generates one even  $X_n$  and one odd  $X_n$ ,

$$I = I_1 + I_2,$$

where

$$\begin{aligned} I_1 &= q(0|n \text{ odd}) \log 2\pi + q(1|n \text{ odd}) \log(2(1-\pi)) \\ &= \pi \log(2\pi) + (1-\pi) \log(2(1-\pi)), \end{aligned}$$

and

$$\begin{aligned} I_2 &= q(0|n \text{ even}) (-\log 2\pi) + q(1|n \text{ even}) (-\log(2(1-\pi))) \\ &= -1/2 \log(4\pi(1-\pi)). \end{aligned}$$

Table 1 shows the dependence of the information numbers on  $P_e$ , the symbol error probability. It should be noted that the information numbers decrease substantially as the symbol error probability becomes larger. Thus, the ARL's become less favorable as  $p_e$  increases.

If one makes the very slight change of inspecting the counter only at even  $n$ , i.e., once for each pair of symbols, then a simpler description of the counter is possible. This is because two consecutive  $X_n$ 's = 0 (successful parity checks) yield a net change of zero, as do consecutive  $X_n$ 's = 1. If, however, one of the pair of  $X_n$ 's is 0 and the other 1, then the total increment of the counter is easily seen to be  $\pm \log \pi/(1-\pi)$ , with + if and only if the odd  $X_n = 0$  (i.e., both a successful check for the out-of-sync hypothesis and a failure for the in-sync hypothesis). Thus the counter moves up and down by a fixed step size and standard random walk formulas (Ref. 7, p. 351) can be used to derive exact formulas for the ARL's under the simplifying assumption of independence. Assuming without loss of generality that

$m$  an integer, one has

$$\text{short ARL} = (\pi - 0.5)^{-1} \left( m + \frac{1 - [\pi/(1-\pi)]^{-m}}{1 - [\pi/(1-\pi)]} \right)$$

and

$$\text{long ARL} = (\pi - 0.5)^{-1} \left( \frac{[\pi/(1-\pi)]^m - 1}{1 - (\pi - 0.5)/\pi} - m \right)$$

Table 2 illustrates the relationship between the two ARL's as a function of  $m$  for two symbol error probabilities.

### A. Counter with Memory

A simple parity check counter does a fairly good job of node synchronization in case of high SNR (Ref. 5). But in the case of high symbol error probability, the probability of parity check error is quite high. For example, symbol error probability 0.1 corresponds to  $\pi = 0.45$ . Thus, in-sync data with  $p_e = 0.1$  will fail an even parity check with probability 0.45, while out-of sync data will fail the even parity checks with probability 0.5. A counter to distinguish between distributions which are so close will either require a long time to react to incorrect sync or have a large probability of incorrect change. Performance numbers for such a counter are indicated in Table 2.

If the parity checks were independent, there would be no way to improve this performance (Ref. 6). But parity checks are not independent. This is because, under the in-sync hypothesis, one channel symbol error changes the value of five even parity checks. Under the in-sync hypothesis, for example, an isolated error in the  $(n-12)$ th channel symbol will cause parity check failures at time  $n, n-2, n-6, n-8$ , and  $n-12$ . Thus a long sequence of successful parity checks would lead us to believe that another success is on the way, while the sequence  $X_{n-12} = 1, X_{n-10} = 0, X_{n-8} = 1, X_{n-6} = 1, X_{n-4} = 0, X_{n-2} = 1$  would lead one to believe that  $X_n$  is very likely to be 1. The reason a log-likelihood counter works so well in the independent case is that by adding logarithms we multiply their arguments. After receiving parity checks  $X_1 = x_1, X_2 = x_2, \dots, X_n = x_n$ , where each  $x_i$  is zero or one, the counter contains

$$\log \frac{q(x_1)}{p(x_1)} + \log \frac{q(x_2)}{p(x_2)} + \dots + \log \frac{q(x_n)}{p(x_n)} = \log \frac{\prod_{i=1}^n q(x_i)}{\prod_{i=1}^n p(x_i)}.$$

If the parity checks were independent, this would be exactly

$$\log \left( \frac{q(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)}{p(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)} \right),$$

the log of the ratio of the likelihood of the string

$$(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$$

under the two hypotheses, which is statistically optimal for detecting loss of synchronization.

In the case of noisy convolutionally encoded data, the probability of a string of parity checks is not just the product of the probabilities of the individual parity checks. Updating the  $p$  probability of a string requires the conditional probability  $p(X_n = x_n | X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, \dots, X_1 = x_1)$ . Therefore, a counter with increment

$$\log \left( \frac{q(X_n = x_n | X_{n-1} = x_{n-1}, \dots, X_1 = x_1)}{p(X_n = x_n | X_{n-1} = x_{n-1}, \dots, X_1 = x_1)} \right)$$

would, after step  $r$ , contain

$$\sum_{n=1}^r \log \frac{q(X_n = x_n | X_{n-1} = x_{n-1}, \dots, X_1 = x_1)}{p(X_n = x_n | X_{n-1} = x_{n-1}, \dots, X_1 = x_1)} =$$

$$\log \frac{\prod_{n=1}^r q(X_n = x_n | X_{n-1} = x_{n-1}, \dots, X_1 = x_1)}{\prod_{n=1}^r p(X_n = x_n | X_{n-1} = x_{n-1}, \dots, X_1 = x_1)} =$$

$$\log \frac{q(X_r = x_r, X_{r-1} = x_{r-1}, \dots, X_1 = x_1)}{p(X_r = x_r, X_{r-1} = x_{r-1}, \dots, X_1 = x_1)},$$

exactly the log of the quantity we desire. Of course, a real counter won't take into account a past of indefinite length. But a (possibly large) integer  $m$  can be chosen, and a counter constructed whose increment at time  $n$  is

$$\log \left( \frac{q(X_n = x_n | X_{n-1} = x_{n-1}, \dots, X_{n-m} = x_{n-m})}{p(X_n = x_n | X_{n-1} = x_{n-1}, \dots, X_{n-m} = x_{n-m})} \right) \quad (8)$$

As might be expected, for large  $m$ , the information number approaches the information number for the hypothetical counter based on the indefinite past of the parity check sequence. This is verified in Appendix A. Moreover, the information numbers obtained using the parity check sequence are identical with information numbers obtainable from the hard quantized received sequence. This means that there is no loss of information or efficiency in using the parity check sequence instead of the hard quantized received sequence to detect loss of node synchronization. This is also shown in Appendix A. Also, Appendix A shows that the counter increments depend only on parity checks of the same (odd or even) type, i.e., (8) is unchanged if the given  $(X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, \dots)$  is replaced by  $(X_{n-2} = x_{n-2}, X_{n-4} = x_{n-4}, \dots)$ .

We will describe the counter in terms of the number of parity checks used to determine the counter increment, and we will call this  $k$ . For example, the "simple" (memoryless) parity check counter corresponds to  $k = 1$ , while the counter for  $k = 8$  has counter increments

$$\log \left( \frac{q(X_n = x_n | X_{n-2} = x_{n-2}, \dots, X_{n-14} = x_{n-14})}{p(X_n = x_n | X_{n-2} = x_{n-2}, \dots, X_{n-14} = x_{n-14})} \right)$$

In general, the counter using  $k$  parity checks has increment

$$\log \left( \frac{q(X_n = x_n | X_{n-2} = x_{n-2}, \dots, X_{n-2k+2} = x_{n-2k+2})}{p(X_n = x_n | X_{n-2} = x_{n-2}, \dots, X_{n-2k+2} = x_{n-2k+2})} \right)$$

The system which we have investigated in detail is therefore a system which takes hard-quantized received channel symbols  $R_1, R_2, \dots$ , creates parity checks  $X_{14}, X_{15}, \dots$ , with

$$\begin{aligned} X_i &= R_i + R_{i-1} + R_{i-3} + R_{i-4} + R_{i-5} + R_{i-6} \\ &\quad + R_{i-7} + R_{i-10} + R_{i-12} + R_{i-13}, \end{aligned}$$

and keeps a counter whose increment at time  $n$  is

$$\log \left( \frac{q(X_n = x_n | X_{n-2} = x_{n-2}, X_{n-4} = x_{n-4}, \dots, X_{n-2k+2} = x_{n-2k+2})}{p(X_n = x_n | X_{n-2} = x_{n-2}, X_{n-4} = x_{n-4}, \dots, X_{n-2k+2} = x_{n-2k+2})} \right).$$

In order to design this counter, we need to know

$$p(X_n = x_n | X_{n-2} = x_{n-2}, \dots, X_{n-2k+2} = x_{n-2k+2})$$

and  $q$  of the same event for even and odd  $n$ . To calculate these values, remember that for in-sync data, the even parity checks depend only on the sequence of channel symbols errors. So given a probability  $p_e$  of symbols error, we can calculate the probability that

$$E_n = e_n, E_{n-1} = e_{n-1}, \dots, E_{n-2k-1} = e_{n-2k-1}$$

for each sequence  $e_n, e_{n-1}, \dots, e_{n-2k-1}$ , and use these probabilities to calculate

$$p(X_n = x_n, \dots, X_{n-2k+2} = x_{n-2k+2})$$

and

$$p(X_{n-2} = x_{n-2}, \dots, X_{n-2k+2} = x_{n-2k+2})$$

and find

$$p(X_n = x_n | X_{n-2} = x_{n-2}, \dots, X_{n-2k+2} = x_{n-2k+2})$$

for even  $n$ .

For odd  $n$ , just as in the case of the simple counter, the  $X_n$ 's are independent with  $p(X_n = 0) = 1/2$ . Thus

$$p(X_n = x_n | X_{n-2} = x_{n-2}, \dots, X_{n-2k+2} = x_{n-2k+2}) = 1/2$$

for odd  $n$ .

To calculate the probabilities  $q$  associated with the out-of-sync hypothesis, just exchange even with odd in the above argument. From these values, we can calculate the counter increments. We did this, assuming a channel symbol error rate of 0.1, which corresponds to a Viterbi decoded bit error rate of  $5 \times 10^{-3}$ , the standard for imaging data. Voyager's data rate has been adjusted so that this is the largest channel symbol error rate which will be encountered.

## B. Information Numbers and Run Lengths

Just as in the case of the up-down counter, the average run lengths are essentially determined by the threshold  $\gamma$  and the information number  $I$ , i.e., the average increment of the counter when the data are truly out-of-sync. For large  $\gamma$  the run lengths are approximately

$$\text{short ARL} \approx \frac{\gamma}{I}$$

and

$$\text{long ARL} \approx C e^{D\gamma},$$

where  $C$ ,  $D$ , and  $I$  depend, of course, on  $k$ , the number of parity checks used in determining the counter increment. These approximations were borne out by the simulations reported in Section III, and the constants  $C$  and  $D$  were determined empirically for each  $k$  considered.  $I$ , the information number, is the average increment of the counter when the data are truly out-of-sync. (Of course,  $I$  depends upon  $k$ , the number of parity checks used in determining the counter increment, and is an increasing function of it.)

The information  $I$  from a pair of parity checks is the sum of the information number  $I_1$  from the odd parity checks (the average increment of the counter at odd  $n$ ) and  $I_2$  from the even parity checks. Table 3 shows the contributions of these two parity check subsequences, revealing that the odd checks contribute roughly two-thirds of the total information.

In designing the detection algorithm, we must choose a symbol error probability  $p_e$ , at which the counter is designed to operate most efficiently. In practice, as the true symbol error probability varies with the signal-to-noise ratio, it will generally be smaller than  $p_e$ , so that fewer parity check failures occur. In this case, the counter will perform better whether the data is in sync or not; i.e., the short ARL will be reduced and the long ARL increased.

In case the signal-to-noise ratio is degraded so much that the symbol error probability exceeds  $p_e$ , however, a problem arises in the performance of the counter, as both the short and long ARL's become less favorable. In real life, the symbol error probability is not constant, and the fact that the long ARL's shorten during periods of low signal-to-noise ratio would introduce the same spurious loss of sync which seems to plague the current Viterbi decoders. If the short ARL gets longer during a period of high symbol error rate, this causes the response time to a loss of sync to increase and may cause a string of data to be lost if the short run hits at the time of low signal-to-noise ratio, but a shortening of the long ARL whenever there is such a period will have a far greater effect on the overall behavior of the system.

This degradation of long ARL can be totally eliminated by giving up the information  $I_2$  from the even parity check subsequence. When the counter uses only the odd subsequence, the long ARL is unaffected by the signal-to-noise ratio. This is because the odd checks are completely random (independent, with 50% probability of success) whenever the process is in sync.

Since most of the information comes from the odd subsequence anyway, we believe it is prudent to base the counter on this subsequence alone. The performance parameters in the next section were all calculated for counters based solely on the out-of-sync parity checks.

### III. Performance Numbers

The size of the ROM needed for counter increments is  $2^k$ . As before, we will describe the counter in terms of the number  $k$  of parity checks needed to compute the counter increment.

Once  $k$  is chosen, the log-likelihood scheme determines the counter increments. The only question left in algorithm design is the threshold at which the system is declared out-of-sync. If the threshold is high, the probability of false loss of sync is low, or, equivalently, the time between false losses of sync is long. On the other hand, a high threshold will also make the short run length (or the time between loss of sync and detection of that loss of sync) large.

We first consider the influence of short run length on system performance. There is an obvious reason to want the short run length to be small: the sooner after a loss of sync that the system gets back on track, the better. But there is another reason as well. Voyager has a concatenated coding scheme: after the convolutional code is Viterbi decoded, an additional code, the Reed-Solomon code, is decoded. As far as the Reed-Solomon decoder is concerned, the data stream during the short run is just a stream of bad data. (Of course, if the out-of-sync condition was caused by the deletion of a symbol, and the node-synchronizer solves the problem by deleting another symbol, then a whole bit has been deleted, and frame boundaries are lost as well. In this case, the data during the short run can never be recovered. So we will consider the case in which the total number of channel symbols has not been changed. This will be true when the out-of-sync condition was caused by a spurious loss of sync caused by the node synchronizer, since the synchronizer will alternate adding and deleting channel symbols, and will be true half of the time anyway.) The decoder can recover a fair amount of bad data, and so if the short run is short enough, the Reed-Solomon decoder will be

able to recover it most of the time. So the length of the average short run is not so important as the probability that the Reed-Solomon decoder will be able to recover the data in the short run. For the  $k = 8$  counter, assuming Viterbi burst statistics for 2.3 dB (Ref. 8) and depth 4 interleaved Reed-Solomon words, Table 4 shows the probability of decoding for a word in a frame wholly containing a short run.

This same table shows the long run lengths (both in bits and in time, at the reasonable Voyager data rate of 20,000 bps) for these same thresholds. Looking, for example, at threshold 15, we see that the probability that a word contained in a short run will be corrected by the Reed-Solomon decoder is  $2/3$ , and that sync is lost incorrectly every two days.

Can we do better? In fact, by going to  $k = 16$ , we can do much better. Table 5 shows this same information with  $k = 16$ . With  $k = 16$  and threshold 14.5, a word which is in a frame attacked by a short run will decode correctly with probability 0.86, and the mean time between false losses of sync (average long run time) is 6.7 years.

These numbers were obtained by simulation methods explained in Appendix B.

Several other questions can be asked about the performance of the counter. What if a short run hits more than one frame? With  $k = 16$  and threshold 14.5, the probability that a short run hits more than one frame is 0.033. And even if the short run does intersect more than one frame, the probability that it causes a word error in each frame is less than 0.005. Thus, the probability of a decoder error in each of two consecutive frames because of a spurious loss of node synchronization is less than 0.0002.

Another question is the probability of *some* loss of data due to a short run. We saw that in the case  $k = 16$ , threshold 14.5, the probability that any one word fails to decode is 0.14, but since decoding failures in the four words are by no means independent, this does not tell us the probability that there is some loss of data—that is, that one or more of the four interleaved Reed-Solomon words fails to decode. In the case  $k = 16$ , threshold 14.5, this probability is 0.19.

## References

1. Deutsch, L. J., and Miller, R. L., "The Effects of Viterbi Decoder Node Synchronization Losses on the Telemetry Receiving System," *TDA Progress Report 42-68*, Jet Propulsion Laboratory, Pasadena, Calif., Aug. 15, 1981.
2. Deutsch, L. J., and Miller, R. L., "Viterbi Decoder Node Synchronization Losses in the Reed-Solomon/Viterbi Concatenated Channel," *TDA Progress Report 42-71*, Jet Propulsion Laboratory, Pasadena, Calif., Nov. 15, 1982.
3. Liu, K. Y., and Lee, J. J., "An Experimental Study of the Concatenated Reed-Solomon/Viterbi Channel Coding System Performance and Its Impact on Space Communications," Publication 81-58, Jet Propulsion Laboratory, Pasadena, Calif., Aug. 15, 1981.
4. Golomb, et al., *Digital Communications with Space Applications*, Prentice Hall, 1964, Chapter 7.
5. Greenhall, C. A., and Miller, R. L., "Design of a Quick-Look Decoder for the DSN (7,1/2) Convolutional Code," *DSN Progress Report 42-53*, Jet Propulsion Laboratory, Pasadena, Calif., Oct. 15, 1979.
6. Lorden, G., "Procedures for Reacting to a Change in Distribution," *Annals of Mathematical Statistics*, 42, No. 6, 1897-1908, 1971.
7. Feller, W., *An Introduction to Probability Theory and Its Applications*, John Wiley and Sons, 1968.
8. Miller, R. L., Deutsch, L. J., and Butman, S. A., "On the Error Statistics of Viterbi Decoding and the Performance of Concatenated Codes," Publication 81-9, Jet Propulsion Laboratory, Pasadena, Calif., Sept. 1, 1981.
9. Siegmund, D., "Importance Sampling in the Monte Carlo Study of Sequential Tests," *Annals of Statistics*, 4, 673-684, 1976.

**Table 1. Dependence of information numbers on symbol error probability**

$p$	$\pi$	$I_1$	$I_2$	$I$
0.08	0.5875	0.0154	0.0155	0.0309
0.10	0.5537	$5.78 \times 10^{-3}$	$5.80 \times 10^{-3}$	0.0116
0.12	0.5321	$2.07 \times 10^{-3}$	$2.07 \times 10^{-3}$	$4.14 \times 10^{-3}$

**Table 2. Average run lengths for various thresholds**

$p$	$\pi$	$I$	$m$	Short ARL	Long ARL
0.1	0.5537	0.01157	10	118	547
			15	205	2,062
			20	296	6,693
0.08	0.5875	0.03091	10	88	1,164
			15	145	7,496
			20	202	44,850

**Table 3. Information numbers**

$k$	Total	Out-of-sync	In-sync
16	0.0862	0.0601	0.0261
8	0.0633	0.0406	0.0228

**Table 4. Counter performance with  $k=8$** 

Threshold	Mean short run, bits	Probability that a Reed-Solomon word in a short run will decode	Mean long run	
			Bits	Time
5	116.6	0.98	$6.1 \times 10^4$	3 seconds
10	238.6	0.86	$5.2 \times 10^7$	43 minutes
15	363.3	0.66	$4.5 \times 10^{10}$	26 days
16	387.6	0.62	$1.7 \times 10^{11}$	100 days
18	438.0	0.52	$2.6 \times 10^{12}$	4 years
20	487.4	0.43	$3.8 \times 10^{13}$	60 years

**Table 5. Counter performance with  $k=16$** 

Threshold	Mean short run, bits	Probability that a Reed-Solomon word in a short run will decode	Mean long run	
			Bits	Time
5	88.6	0.99	$5.9 \times 10^6$	5 minutes
10	171.4	0.95	$7.9 \times 10^9$	4 days
11.5	196.4	0.93	$6.0 \times 10^{10}$	35 days
13	221.7	0.90	$5.0 \times 10^{11}$	290 days
14.5	246.9	0.87	$4.2 \times 10^{12}$	6.7 years
18	305.1	0.77	$6.1 \times 10^{14}$	960 years
20	338.8	0.71	$1.0 \times 10^{16}$	16,000 years

## Appendix A

### Proofs

This appendix gives mathematical proofs of the statements made in Section II.

We use the same random processes to model the situation: (1)  $\dots, M_{-2}, M_0, M_2, \dots$  an i.i.d. process,  $P(M_0 = 0) = P(M_0 = 1) = 1/2$ , representing the message; (2)  $\dots, E_{-2}, E_{-1}, E_0, E_1, \dots$  an i.i.d. process,  $P(E_0 = 1) =$  symbol error probability, representing the errors in the received sequence and independent of the sequence  $\dots, M_{-2}, M_0, M_1, \dots$ ; (3)  $\dots, R_{-2}, R_{-1}, R_0, R_1, \dots$ , the convolutionally encoded  $M_i$ 's added to the  $E_i$ 's, representing the hard-quantized received channel symbols; and (4)  $\dots, X_{-1}, X_0, X_1, \dots$  the sequence of parity checks derived from the  $R_i$ 's.

*Proposition:*  $p(X_n = x_n, X_{n-1} = x_{n-1}, \dots, X_{n-2k} = x_{n-2k}) = [p(X_n = x_n, X_{n-2} = x_{n-2}, \dots, X_{n-2k} = x_{n-2k}) \cdot p(X_{n-1} = x_{n-1}, \dots, X_{n-2k+1} = x_{n-2k+1})]$ .

*Proof:* First observe that for  $m$  odd,

$$p(X_m = 1 | X_{m-2}, X_{m-4}, \dots, E_{-1}, E_0, E_1, \dots) = 1/2,$$

because  $X_m$  is a sum of  $M_{m+1}$  and other variables,  $p(M_{m+1} = 1) = 1/2$ , and  $M_{m+1}$  is independent of all the random variables on which we are conditioning. This means that for odd  $m$

$$p(X_m = x_m, X_{m-2} = x_{m-2}, \dots,$$

$$X_{m-2\ell} = x_{m-2\ell} | \dots, E_{-1}, E_0, E_1, \dots) = 2^{-\ell-1} \quad \text{a.s.}$$

for any sequence  $(x_m, x_{m-2}, \dots, x_{m-2\ell})$  of zeroes and ones. But the values of the even parity checks are determined entirely by  $\dots, E_{-1}, E_0, E_1, \dots$ , and so are independent of the odd parity checks.

Notice: (1) The fact that we looked at a sequence of odd length was convenient for notation but had no effect on the proof. (2) We showed not only that the even and odd parity checks are independent, but that the odd parity checks are themselves i.i.d. with probability 1/2 of success. (3) The same result holds for the measure  $q$ , reversing the roles of even and odd.

*Corollary:*  $p(X_n = x_n | X_{n-1}, X_{n-2}, \dots) = p(X_n = x_n | X_{n-2}, X_{n-4}, \dots)$ , and the same for  $q$ . If  $n$  is odd,  $p(X_n = x_n | X_{n-2}, X_{n-4}, \dots) = 1/2$ ; for even  $n$   $q(X_n = x_n | X_{n-2}, \dots) = 1/2$ .

*Definition:* The one-sided, pseudo parity check sequence  $\bar{X}_1, \bar{X}_2, \dots$  is the parity check sequence based on the process  $\dots, 0, 0, R_1, R_2, R_3, \dots$ . That is,  $\bar{X}_1 = R_1, \bar{X}_2 = R_2 + R_1, \bar{X}_3 = R_3 + R_2, \dots, \bar{X}_n = X_n$  for  $n \geq 14$ .

*Proposition:* Every event of the form  $(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$  corresponds to exactly one event  $(R_1 = r_1, \dots, R_n = r_n)$ .

*Proof:* The  $X_i$ 's are derived from the  $R_i$ 's. Going backwards, knowing  $X_1$  tells you  $R_1$ , and knowing  $X_i$  and  $R_1, \dots, R_{i-1}$  tells you  $R_i$ .

*Corollary:* A log-likelihood counter based on the  $X_i$ 's will always contain exactly the same value as one based on the  $R_i$ 's.

*Proposition:* If  $I_m$  is the information number of any log-likelihood counter with inputs based on the last  $m$  outputs of any discrete random process and  $I$  the information number with increments based on the indefinite past, then  $\lim_{m \rightarrow \infty} I_m = I$ .

*Proof:* For simplicity, we give the proof for random variables taking on the values zero and one. The proof for random variables taking on finitely many values is exactly the same.

Let  $\dots, Y_{-n}, \dots, Y_0, Y_1, \dots$  be the stationary process. (In our case, the  $Y_i$ 's are pairs of hard-quantized received channel symbols). Let

$$f_n(y, j) = \log \left( \frac{[p(Y_0 = j | Y_{-1}, Y_{-2}, \dots, Y_{-n})](y)}{[q(Y_0 = j | Y_{-1}, Y_{-2}, \dots, Y_{-n})](y)} \right),$$

where  $y$  is chosen from the probability space on which the  $Y_i$ 's are defined and  $j$  is zero or one.

It is a standard result of Martingale theory that

$$\lim_{n \rightarrow \infty} f_n(y, j) = f(y, j)$$

$$= \log \left( \frac{p(Y_0 = j | Y_{-1}, \dots)(y)}{q(Y_0 = j | \dots)(y)} \right), \text{ a.s.,}$$

and so by the dominated convergence theorem

$$\lim_{n \rightarrow \infty} \int f_n [y, Y_0(y)] dp = \int f [y, Y_0(y)] dp,$$

and the same for integration with respect to  $q$ . But these are just the information numbers for the counters.

Notice: A log-likelihood counter using  $k$  parity checks based on parity checks  $\bar{X}$  will, after time  $2k + 14$ , have exactly the same increments as a log-likelihood counter using  $k$  parity checks and based on parity checks  $X$ , because the values of  $\bar{X}$  and  $X$  are exactly the same starting at time 14.

*Theorem:* As  $k$  approaches infinity, the information in the log-likelihood counter whose increment at time  $n$  is

$$\log \frac{p(X_n = x_n | X_{n-2}, \dots, X_{n-2k+2})}{q(X_n = x_n | X_{n-2}, \dots, X_{n-2k+2})},$$

kept separately for  $n$  even and odd, approaches all the sync information in the hard-quantized received channel symbol stream  $R_1, R_2, \dots$

*Proof:* As  $m$  goes to infinity, the information in a counter  $C(1)$  based on past of length  $2k$  of the hard-quantized received channel stream approaches all the information in the stream. The values in  $C(1)$  are exactly the same as would be in a counter  $C(2)$  based on the past of length  $2k$  of the (nonstationary) parity checks  $\bar{X}_i$ . For  $n > 2k + 14$ , the increments in counter  $C(2)$  are the same as those in a counter  $C(3)$  based on the probabilities of  $(X_n | X_{n-1}, \dots, X_{n-2k})$ . But, since even and odd parity checks are independent, this is exactly the same as the counter of the theorem.

## Appendix B

### Simulation of ARL's

The performance figures of Section III were obtained by simulation. The short ARL's were simulated directly by generating independent symbol errors with probability  $p$ , computing the resulting parity check stream, and feeding it to the counter with threshold  $\gamma$ . Direct simulation of the long ARL's is not feasible, however, because (as the results show) the time required to generate a statistically useful sample of long run lengths would be too great.

The long ARL's were simulated by a modification of a standard technique called "importance sampling" (Ref. 9), in which the process to be analyzed — in this case, the parity check sequence — is generated using a probability distribution  $Q$  different from the distribution  $P$  for which results are desired. In our case,  $P$  specifies independent 50-50 results for the out-of-sync parity checks — so that the time for the counter to reach a distant threshold  $\gamma$  is quite large. A distribution  $Q$  was chosen to make the counter reach the threshold more quickly — namely, independent parity checks with probability of failure  $\pi^*$ , substantially less than 1/2.

The method of importance sampling is based on the simple fact that for any event,  $A$ , the probability of  $A$  under  $P$  can be obtained from simulations carried out under the distribution  $Q$ . The key is provided by the identity

$$P(A) = \int_A dP = \int_A \left( \frac{dP}{dQ} \right) dQ. \quad (\text{B-1})$$

Here the quantity  $dP/dQ$  is the Radon-Nikodym derivative of  $P$  with respect to  $Q$ , which in our application is simply the likelihood ratio

$$\frac{P(X_1) \cdots P(X_n)}{Q(X_1) \cdots Q(X_n)}$$

of the parity checks  $X_1, \dots, X_n$  up to the time that  $A$  occurs. Since  $P(X_i) \equiv 1/2$ , and  $Q(X_i) = \pi^*$  if  $X_i = 1$  (failure),  $= 1 - \pi^*$  if  $X_i = 0$  (success), relation (1) can be made more explicit. Using  $E_Q$  to denote expectation under  $Q$ , it takes the form

$$P(A) = E_Q \left[ \left( \frac{1}{2\pi^*} \right)^F \left( \frac{1}{2(1-\pi^*)} \right)^S 1\{A\} \right], \quad (\text{B-2})$$

where  $F$  and  $S$  are the numbers of failures and successes, respectively, in the out-of-sync parity checks up to the time that  $A$  occurs, and  $1\{A\} = 1$  if  $A$  occurs,  $= 0$ , otherwise.

The simulation of the long ARL was based on the definition of a *counter cycle*: Starting from a given state  $s^*$  of  $k$  zeroes and ones, the cycle ends the first time that the counter resets to zero with the same sequence  $s^*$  in its memory. Let  $T$  denote the time (number of symbol pairs) for a cycle to end and let  $N$  denote the number of cycles until the threshold  $\gamma$  is crossed. Then using a standard result called Wald's equation (Ref. 7, Vol. II, p. 567), we have

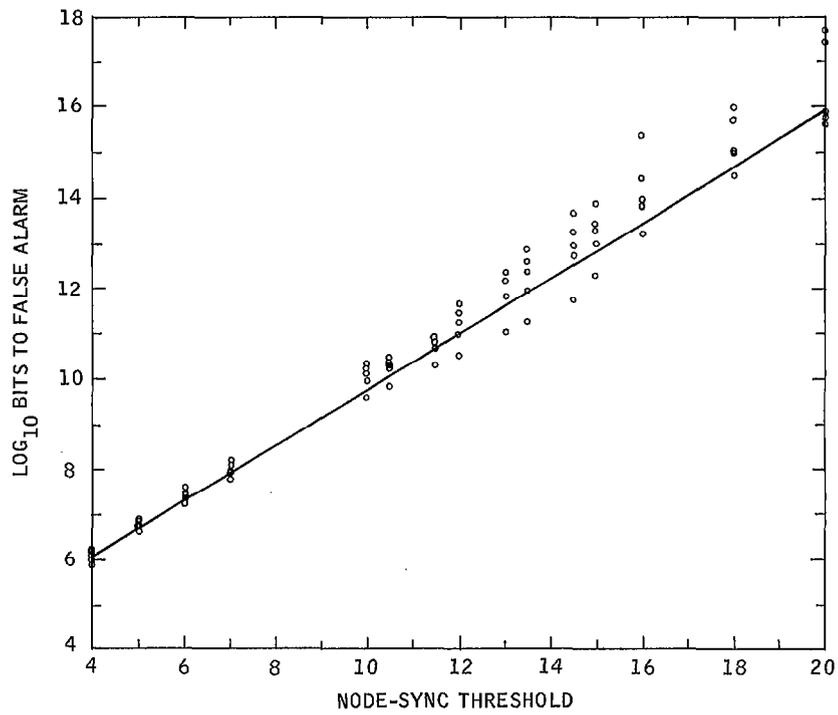
$$\text{long ARL} = EN \cdot ET.$$

(Actually, the right side gives the expected time until the *end* of the first cycle on which  $\gamma$  is crossed, but the extra time to end the cycle after crossing is negligible compared to the long ARL.) The quantity  $ET$  was easily simulated directly, since when the parity checks are random the cycles end fairly quickly (and don't depend on  $\gamma$  at all). The evaluation of  $EN$  was based on

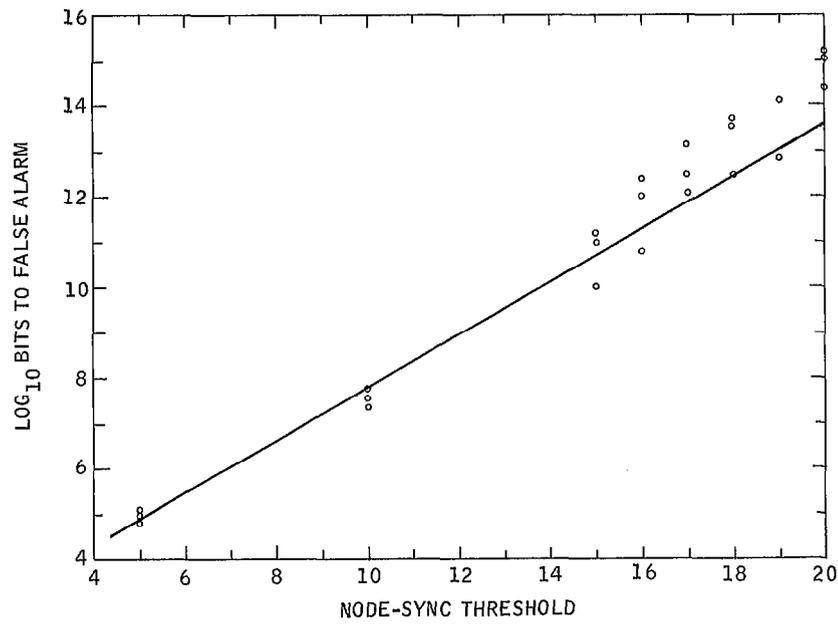
$$EN = \frac{1}{P(A)},$$

where  $A = \{\gamma \text{ is crossed}\}$  for a given cycle, and  $P(A)$  was simulated using the method of importance sampling, as described above.

Importance sampling was used to estimate  $P(A)$  in several independent sets of simulations. Not surprisingly, the estimates were more stable for smaller thresholds. After all the data were gathered, a least squares line was drawn through the points representing small thresholds (see Figs. B-1 and B-2). These lines were used for the long ARL's in Tables 4 and 5.



**Fig. B-1. Mean time to false alarm,  $k=16$ ; least squares line from thresholds 4, 5, 6, and 7**



**Fig. B-2. Mean time to false alarm,  $k=8$ ; least squares line from thresholds 5, 10, and 15**