

Frame Synchronization Without Attached Sync Markers

Jon Hamkins
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099
Jon.Hamkins@jpl.nasa.gov

Abstract—We describe a method to synchronize codeword frames without making use of attached synchronization markers (ASMs). Instead, the synchronizer identifies the code structure present in the received symbols, by operating the decoder for a handful of iterations at each possible symbol offset and forming an appropriate metric. This method is computationally more complex and doesn't perform as well as frame synchronizers that utilize an ASM; nevertheless, the new synchronizer acquires frame synchronization in about two seconds when using a 600 kbps software decoder, and would take about 15 milliseconds on prototype hardware. It also eliminates the need for the ASMs, which is an attractive feature for short uplink codes whose coding gain would be diminished by the overhead of ASM bits. The lack of ASMs also would simplify clock distribution for the AR4JA low-density parity-check (LDPC) codes and adds a small amount to the coding gain as well (up to 0.2 dB).

TABLE OF CONTENTS

1 INTRODUCTION	1
2 PRELIMINARIES	1
3 AN ASM-LESS FRAME SYNCHRONIZER FOR LDPC CODES	3
4 PERFORMANCE	4
APPENDIX: MAXIMUM LIKELIHOOD SYNCHRONIZER	6
ACKNOWLEDGMENTS	7
REFERENCES	7
BIOGRAPHY	7

1. INTRODUCTION

The emerging Consultative Committee for Space Data Systems (CCSDS) recommendation for low-density parity-check (LDPC) codes [2] specifies a 64-bit attached sync marker (ASM) that is to immediately precede each LDPC codeword. The codeword (frame) boundary can be identified by searching for the ASM.³ In an argmax-type synchronizer, a metric is computed for each candidate offset, and the offset with the highest metric is declared the winner. This type of synchronizer has been successfully used for the (2048,1024) LDPC code in tests at the Electronic Systems Test Laboratory

¹ 978-1-4244-7351-9/11/\$26.00 ©2011 IEEE. ² IEEEAC paper # 1703, Version 2, Updated December 29, 2010³ Kenneth Andrews, "Frame synchronizers without (very many) equations," JPL Interoffice Memorandum, November 2007.

(ESTL) at the Johnson Space Center, for example.^{4,5}

A brute force way to synchronize frames is to buffer two frame-lengths of symbols—a length sufficient to guarantee capture of at least one complete frame—and attempt decoding at each possible offset until an offset is found for which decoding is successful. This decode-at-all-offsets approach was used in the Mars Laser Communications Demonstration⁶ (MLCD) for example, and works well if the decoder is many times faster than the data rate of the link. For CCSDS LDPC codes, the decoder would need to operate four to five orders of magnitude faster than the data rate in order for it to be able to acquire the correct frame offset without dropping or buffering additional codewords during the synchronization process. This is because, first, there are $n = 1280$ to 32768 candidate offsets to consider for the CCSDS LDPC codes, depending on the output length of the code; and second, the decoder can be an order of magnitude slower when attempting to decode candidate offsets as it is in its usual decoding operation — a result of the decoder using the maximum (e.g., 200), not average (e.g., 20), number of iterations at each incorrect offset.

In this paper, we present a variation of the brute force approach in which decoding is halted prematurely. At the correct offset, the messages passed in the decoding algorithm begin to converge in a fundamentally different way than they do when the offset is incorrect. This difference can be exploited by forming an appropriate metric that discriminates between the correct offset and the incorrect offsets.

2. PRELIMINARIES

Data flow

This paper assumes a data flow as shown in Figure 1. Information bits are sent to the encoder of an LDPC or turbo code, or any any code which is iteratively decoded with a message-passing algorithm that uses the concept of check nodes or log-likelihood ratios (LLRs) of the transmitted symbols. To remove the potential issue of false frame-synchronization of quasi-cyclic shifts of codewords,⁷ we assume the use of the CCSDS randomizer [1, section 6], which in any case is good

⁴ Kenneth Andrews, "FPGA core: Frame synchronizer, two-marker list-based algorithm," JPL Interoffice Memorandum, July 2007. ⁵ Chatwin Landsdowne, "LM Orion baseband processor 'brassboard II' test – out-brief results summary," October 2009. ⁶ William Farr, "MLCD Ground Network Preliminary Design Review – 1.025 Mbps end-to-end link demonstration," May 2005. ⁷ Kenneth Andrews, "Results of the LDPC decoder tests at ESTL," JPL Interoffice Memorandum, March 2007.

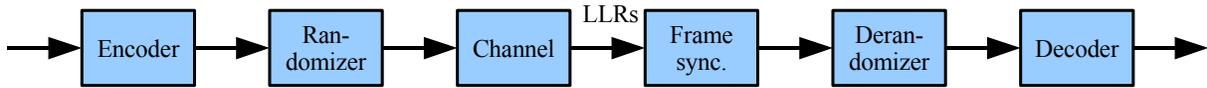


Figure 1. Data flow.

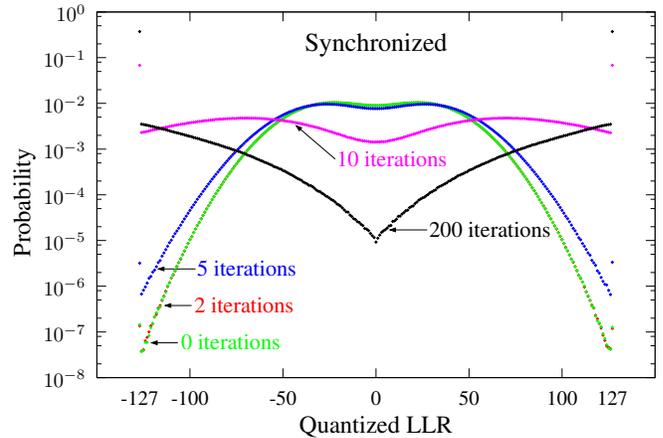
practice when the transition density in the information bits is unknown. The codeword symbols are transmitted contiguously, without gaps. For our purposes, we assume an ASM is not transmitted, although the algorithm would still work if it were – it does not utilize the ASM. At the receiving end, LLRs of the received symbols are formed.

A $2n$ -vector of these LLRs is stored, where n is the output length of the (n, k) code. Thus, a codeword starts somewhere within the first n positions, and the vector contains the complete set of LLRs of a codeword. The goal is to identify which of the first n positions corresponds to the first symbol. After frame synchronization, the LLRs are derandomized by flipping their sign bits as necessary,⁸ and decoding can begin.

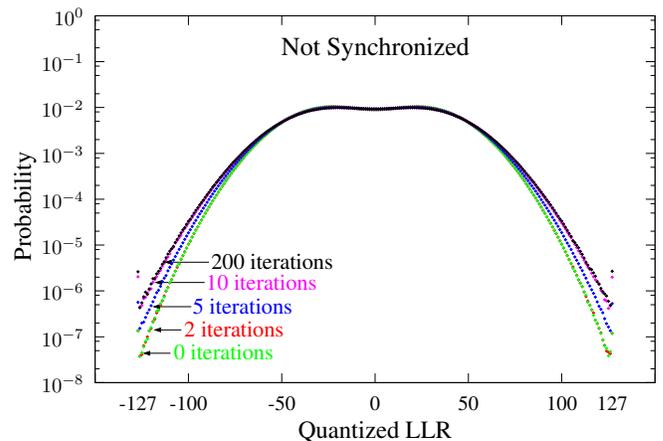
LLR distributions

As messages are passed on the Tanner graph of the code during decoding, the reliabilities of the transmitted symbols will evolve. If the frame is properly synchronized, the reliabilities usually improve as decoding continues, and if a correct codeword emerges, many of the reliabilities will become quite high. If the frame is incorrectly synchronized, the passage of the mis-synchronized LLRs through the derandomizer causes the Tanner graph to be initialized with a set of reliabilities that typically don't relate in any way to the code structure and the nodes don't typically converge to high reliabilities.

This behavior is shown in Figure 2 for the (2048,1024) AR4JA LDPC code. In Figure 2(a), the LLR distributions are shown when the decoder⁹ uses correctly synchronized frames and stops after 0, 2, 5, 10, and 200 iterations. As is seen in the figure, the first two iterations provide no noticeable difference in the LLR distribution, but by the fifth iteration, the LLR distribution is evolving to higher reliabilities. In each case, $E_b/N_0 = 2$ dB, which is the approximate threshold at which the code can be operated effectively. About 14.4 iterations are needed to decode, on average, at that signal-to-noise ratio (SNR). Figure 2(b) shows the LLR distributions when the frame is mis-synchronized by one symbol. The reliabilities increase slightly as the decoder struggles to make sense of its meaningless input, but by about the tenth iteration the reliabilities have stagnated. The LLR reliabilities after five iterations are noticeably worse than the corresponding LLR reliabilities in the synchronized case.



(a) Using correctly synchronized input.



(b) Using incorrectly synchronized input.

Figure 2. LLR distribution, with (2048,1024) AR4JA code at $E_b/N_0 = 2$ dB.

Number of satisfied check nodes

The number of satisfied check nodes will also evolve during decoding. If the frame is properly synchronized, the number of satisfied check nodes usually increases as decoding continues, until all are satisfied and a valid codeword is declared. Decoding fails if the maximum number of allowed iterations is reached before all check nodes are satisfied. If the frame is incorrectly synchronized, the Tanner graph is initialized with a set of variable node LLRs that typically don't relate in any way to the code structure and the count of satisfied check nodes will generally start lower and will fail to reach a large number with continued iterations.

This behavior is shown in Figure 3 for the (2048,1024) AR4JA LDPC code. In Figure 3(a), the distributions of the number of failed check nodes are shown when the decoder

⁸ Kenneth Andrews, "LDPC codes in a protocol stack," JPL Interoffice Memorandum, March 2007.⁹ The software decoder utilized here quantizes the LLRs to integer values ranging from -127 to 127. The quantizer is designed so that in its granular region, the quantized LLR is roughly eight times the LLR.

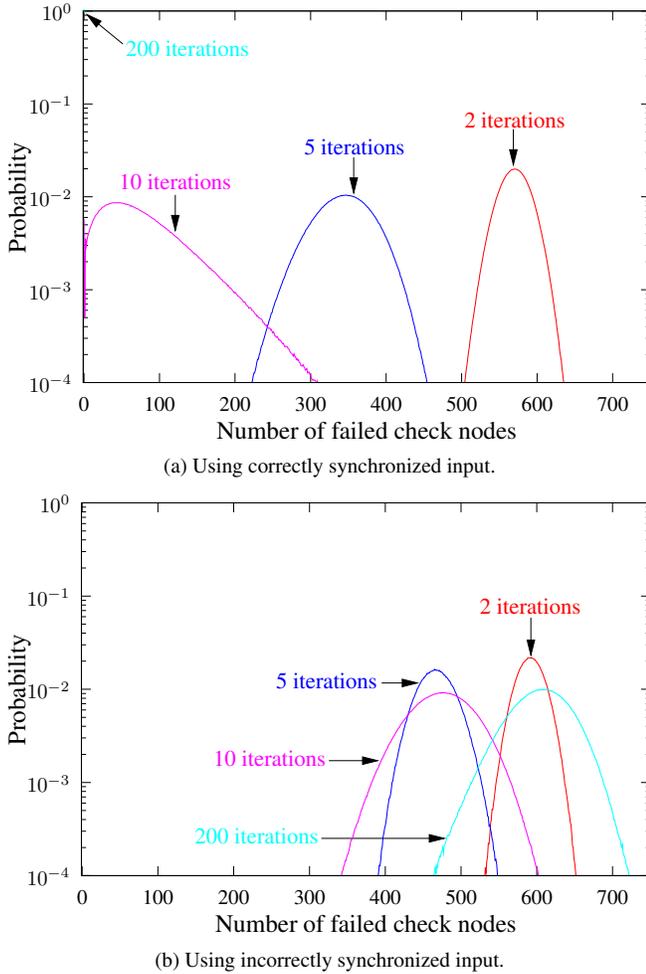


Figure 3. Distribution of number of failed check nodes, with (2048,1024) AR4JA code at $E_b/N_0 = 2$ dB.

uses correctly synchronized frames and stops after 2, 5, 10, and 200 iterations. As is seen in the figure, an increasing number of iterations results in fewer failed check nodes, and in this simulation, all of the codewords were correctly decoded by the 200th iteration. In each case, the (2048,1024) LDPC code was simulated at $E_b/N_0 = 2$ dB.

Figure 3(b) shows the distribution of the failed check node count when the frame is mis-synchronized by one symbol. Initial iterations reduce the number of failed check nodes, until about the fifth iteration. Further iterations actually increase the number of failed check nodes, and the decoder becomes hopelessly lost.

3. AN ASM-LESS FRAME SYNCHRONIZER FOR LDPC CODES

Defining a metric

The decoder may distinguish between the synchronized and un-synchronized conditions by forming metrics based on the variable node LLRs or check node satisfactions produced by a partial decoding. This is conceptually similar to the metric-

growth-rate method of acquiring node synchronization for convolutional codes [3], except that there are many more candidate offsets to consider. A Maximum Likelihood (ML) approach in arriving at a statistic is derived in the Appendix; here, we discuss ad hoc approaches. When variable node LLRs are used to discriminate between synchronized and un-synchronized offsets, one ad hoc metric is

$$M = \sum_{i=1}^n f(\lambda_i) \quad (1)$$

where λ_i is the i th LLR and $f(\cdot)$ is a function to be defined. We may reasonably require $f(\cdot)$ to be an even, monotonically increasing function of its argument, so that it is unbiased with respect to the number of 1s in a codeword and so that it rewards higher reliabilities. Some possibilities for $f(\cdot)$ are:

1. $f(x) = |x|^a$, for some real positive a
2. $f(x) = e^{|x|}$
3. $f(x) = \log(1 + |x|)$
4. $f(x) = I_{\{|x| \geq \eta\}}$, where I is the indicator function and η is a threshold

When check nodes are used to form an ad hoc metric, one choice is the number of satisfied check nodes:

$$M = \sum_{i=1}^{n-k} I_{\{\text{check node } i \text{ satisfied}\}} \quad (2)$$

One could also form metrics using both the variable and check nodes. One could also incorporate the evolving nature of the metrics, e.g., by measuring whether the number of satisfied check nodes is increasing or whether the variable node reliabilities are increasing from one iteration to the next.

Description of Frame Synchronizer algorithm

We can now summarize the steps to synchronize the frames of an (n, k) code:

1. Collect soft symbols (LLRs) $y_0, y_1, \dots, y_{2n-1}$ from the channel
2. For each $j, j = 0, 1, \dots, n-1$,
 - (a) Derandomize $y_j, y_{j+1}, \dots, y_{j+n-1}$
 - (b) Run the decoder for I iterations using the derandomized $y_j, y_{j+1}, \dots, y_{j+n-1}$ as input
 - (c) Form metric $M(j)$ as given in (1) or (2).
3. Declare $j^* = \underset{j}{\operatorname{argmax}} M(j)$ to be the index of the first symbol of the codeword
4. Run the decoder for its maximum number of iterations (e.g., 200) using $y_{j^*}, y_{j^*+1}, \dots, y_{j^*+n-1}$ as input.
 - (a) If decoding fails, this may be the result of frame synchronization error. Repeat the frame synchronizer process at step 1, using a new window of $2n$ channel symbols (including n new symbols), i.e., $y_n, y_{n+1}, \dots, y_{3n-1}$.
 - (b) If decoding succeeds, the frame synchronization has been acquired correctly. END.

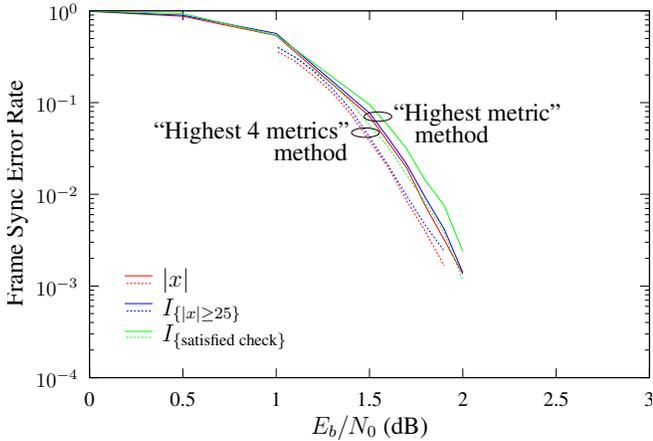


Figure 4. Frame synchronization error rate for various metrics.

If the frame synchronizer is fast enough relative to the incoming symbol rate, it may be run after reception of every codeword, essentially providing continuous synchronization functionality. Otherwise, it may be run once to acquire frame synchronization, and then remain dormant as long as correct decoding ensues. If a previously-designated number of consecutive codewords fail to decode, then the synchronization process may be restarted. In this re-acquisition phase, preference could be given to the several adjacent symbol positions, to minimize the re-acquisition time.

4. PERFORMANCE

Frame Synchronization Error Rate vs. choice of metric.

We simulated the frame synchronizer for the (2048,1024) AR4JA LDPC code operating at $E_b/N_0 = 2$ dB, using variable nodes metrics $f(x) = |x|, \sqrt{|x|}, x^2, x^4, \log(1 + |x|), I_{\{|x| \geq 25\}}, I_{\{|x| \geq 50\}}, I_{\{|x| \geq 75\}}, I_{\{|x| \geq 100\}},$ and check nodes metric $I_{\{\text{satisfied check}\}}$. We found that the synchronizer performance is not sensitive to choice of metric function. This is illustrated, for 10 iterations and a subset of the metrics, in Figure 4. Also shown in the figure is the performance of a simple extension of the synchronizer, in which instead of declaring the offset corresponding to the highest metric to be the correct one, the offsets corresponding to the highest *four* metrics are each decoded using full iterations and the one successfully decoded is declared the correct one. This extension does not significantly improve performance, which indicates that when the synchronizer fails, the correct offset often does not have a competitively high metric. The insensitivity to choice of metric suggests choosing a simple one, such as $|x|$ or $I_{\{\text{satisfied check}\}}$. In the results in the remainder of the paper, we use $|x|$ as the metric.

Frame Synchronization Error Rate vs. number of iterations.

The frame synchronization error rate for the (2048,1024) AR4JA code at $E_b/N_0 = 2$ dB is shown in Figure 5. We desire that the frame synchronizer operate at an SNR below that at which the decoder can successfully decode bits, so that the

synchronizer is not the performance limiter in the end-to-end system. Unfortunately, this requires setting the number of iterations to a fairly high number in the frame synchronization algorithm. As seen in Figure 5(a), 50 iterations are necessary in the frame sync algorithm in order to achieve frame sync error rate performance comparable to the codeword error rate (CWER) of the 200-iteration decoder — this, in spite of the excellent separation we see in Figure 2 in the variable node distributions in as few as 10 iterations. The problem, apparently, is that when thousands of offset metrics are competing, a single outlier that exceeds the correct offset metric is relatively common.

This problem may be overcome by processing more than one codeword, i.e., by exercising step 4a of the algorithm one or more times. A 10 iteration synchronizer achieving frame synchronization error rate 10^{-3} based on processing one codeword, for example, would achieve $(10^{-3})^2 = 10^{-6}$ by processing two codewords, assuming the noise is independent from one codeword to the next. Figure 5(b) shows the frame sync error rate when two codewords are used. At very low data rates, there may be time to perform many iterations, but at higher rates, rather than perform 50 or more decoding iterations for the synchronizer, one could use fewer iterations, e.g., as many as can be used to check all n offsets before the next codeword arrives.

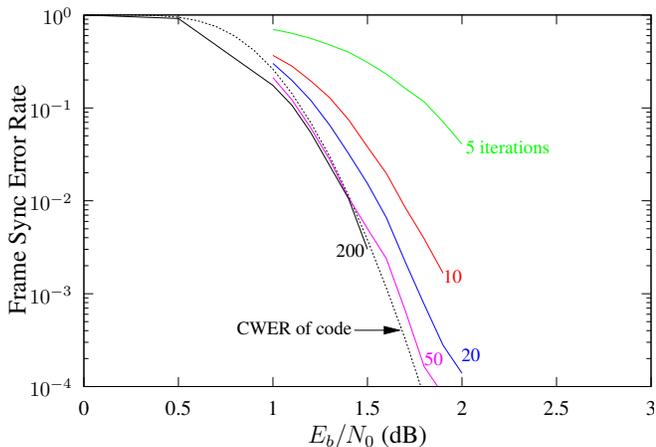
Acquisition Time

In determining the mean time to acquire frame synchronization, we assume that the first n symbols have already arrived at the receiver — this is a necessary condition for virtually any synchronizer to function — and we measure the average time, T_{acq} , it takes for the proposed synchronizer to settle on the correct symbol offset. This includes the time to make computations, as well as the time needed to receive and process additional symbols beyond the first n , if necessary.

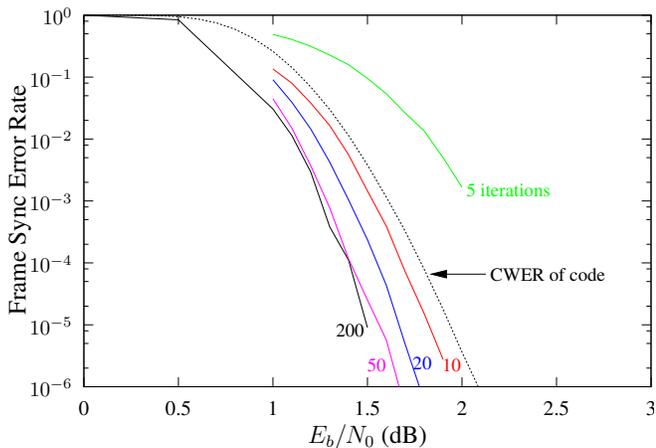
Suppose the synchronizer for an (n, k) code uses I iterations to check each offset, and each iteration takes T_1 s. If $I \geq I_{\max}$, where I_{\max} is the maximum number of iterations used by the decoder in its normal decoding mode, then the synchronizer will find the correct offset after checking at most n offsets — one of them will succeed in finding the codeword (neglecting the small probability of a decoder failure at the proper offset). The decoder completes I iterations in time IT_1 , and if this time is shorter than a symbol duration T_{sym} , the synchronizer must wait for the next symbol to arrive before checking the next offset. Thus, the acquisition time when $I \geq I_{\max}$ is

$$T_{acq} = \max \{ nIT_1, nT_{sym} \}.$$

Now suppose $I < I_{\max}$. Let p_I denote the resulting frame synchronization error rate when based on one codeword, i.e., when the synchronizer computes n metrics. Each sync error takes nIT_1 s (or nT_{sym} , whichever is longer) to compute the synchronization metrics, followed by $I_{\max}T_1$ s to attempt



(a) Based on one codeword.



(b) Based on two codewords.

Figure 5. Frame synchronization error rate as a function of number of iterations performed.

decoding and declare failure after I_{\max} iterations. This is repeated, until (on the j th try) synchronization succeeds. Thus, the mean acquisition time is given by

$$\begin{aligned}
 T_{acq} &= \sum_{j=1}^{\infty} \underbrace{[1 - p_I] p_I^{j-1}}_{\text{Probability sync takes } j \text{ codewords}} \cdot \underbrace{(jn \max(IT_1, T_{sym}) + (j-1)I_{\max}T_1)}_{\text{Acquisition time with } j \text{ codewords}} \\
 &= \frac{n \max(IT_1, T_{sym}) + p_I I_{\max} T_1}{1 - p_I}. \quad (3)
 \end{aligned}$$

The numerator and denominator are both nondecreasing in I , and we may choose I to minimize the acquisition time. If $IT_1 < T_{sym}$, the synchronizer operates faster than symbols arrive from the channel, and one can increase I without increasing T_{acq} .

We now consider a quantitative example. The (2048,1024) AR4JA LDPC code operating at $E_b/N_0 = 2$ dB can be decoded at $R_{dec} = 597$ kbps and $I_{dec} = 14.4$ average iterations using C software running on a standard desktop. Thus, the

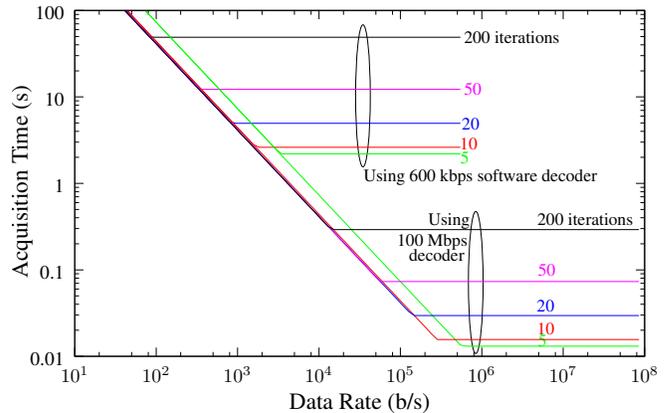


Figure 6. Average acquisition time for AR4JA (2048,1024) code, at $E_b/N_0 = 1.5$ dB.

time to perform one iteration is given by

$$T_1 = \frac{k}{I_{dec} R_{dec}} = \frac{1024}{14.4 \cdot 597000} = 0.12 \text{ ms}.$$

Using the p_I values from Figure 5(a), a maximum of $I_{\max} = 200$ iterations, and a symbol duration of $T_{sym} = 2/R$ s, where R is the data rate of the link in bits per s, we may plug into (3). The result is shown in Figure 6. For data rates above a few kbps, the average acquisition time is about 2 seconds when using a 5-iteration synchronizer. At lower data rates, the synchronizer is limited by the time it takes to receive additional symbols from the channel. The figure also shows the acquisition time when a 100 Mbps decoder is available. Acquisition times range from 0.013 s for the 5-iteration synchronizer to 0.29 s for the 200-iteration synchronizer. These data show that acquisition time is minimized by using only a few decoder iterations, substantially fewer than the number needed to produce decoded bits.

Uplink Codes

The standard CCSDS code used for many years on uplink channels has output length 63, sometimes padded to 64. Its low coding gain makes it a ripe target for replacement by a modern LDPC code. If such an LDPC code also has a very short length, then an ASM of 32 or 64 bits could effectively double the length of the transmission, wiping out a large fraction of the coding gain. This could make selection of LDPC codes unattractive for the uplink application, despite their large coding gain. The elimination of the ASM, then, may enable the consideration of a larger set of candidate codes, leading to increased coding gains on the uplink channel. The increased coding gain would enable NASA to command spacecraft to further reaches of the solar system, or beyond, or to command closer spacecraft using smaller antennas than would otherwise be possible.

Data Volume on the Downlink

To first order, the method of frame synchronization does not affect the total data volume that a link can support on its

downlink channel, but there is a small increase, negligible in most cases, in throughput because of the elimination of the ASMs. This throughput gain is less than 0.01 dB for each of the CCSDS codes, with the exception of the $k = 1024$ AR4JA LDPC codes, where the gain ranges from 0.134 to 0.212 dB.¹⁰ This means that a 1 Mbps link could be improved to 1.05 Mbps, which over the course of an eight hour pass would improve the data volume by 1.36 Gb.

This increase in data volume may be reduced or reversed if frames are lost during acquisition or re-acquisition of frame synchronization. The effect would be more likely at higher data rates, and if link conditions were such that symbol slipping in the receiver were prevalent.

Clock Distribution

Eliminating the ASMs simplifies the relationship between the input and output symbol rates to the encoder for an AR4JA code. The (2048,1024) AR4JA code with the 64-bit marker, for example, takes as input 1024 bits for each 2048+64=2112 symbols at its output. This is an input-output ratio of 16:33. Without the ASM, the input-output ratio is 1:2, which may simplify clock distribution in the baseband processing of the spacecraft. Similarly, exact 2:3 and 4:5 relationships emerge for the 2/3 and 4/5 AR4JA codes.

APPENDIX: MAXIMUM LIKELIHOOD SYNCHRONIZER

The main part of the paper shows several ad hoc metrics that can be used in the development of an ASM-less frame synchronizer. In this appendix, we show that the ad hoc check node metric in (2) follows closely from a derivation of the Maximum Likelihood (ML) statistic for frame synchronization.

Preliminaries

LLR of a check node—To discuss the probability that a check node is satisfied, it is helpful to form a log likelihood ratio (LLR) statistic for it. To start, we'll consider a single check node with degree d . Suppose, for $i = 1, \dots, d$, we have

$$y_i = x_i + n_i,$$

where $x_i \in \{-1, +1\}$ and n_i is a zero-mean Gaussian random with variance σ^2 . We assume n_i is independent of x_j for all i and j , and independent of n_j for $i \neq j$. Define the vectors $\mathbf{x} \triangleq (x_1, \dots, x_d)$ and $\mathbf{y} \triangleq (y_1, \dots, y_d)$. Then the conditional probability density function of \mathbf{y} given \mathbf{x} is

$$f(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^d \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[\frac{-(y_i - x_i)^2}{2\sigma^2} \right].$$

Since \mathbf{x} takes values in $\{-1, +1\}$, we may write $1 - 2\mathbf{x}$ when we wish to refer to values in $\{0, 1\}$. Let $h(\mathbf{x}) \triangleq (1 - 2x_1) \oplus$

$(1 - 2x_2) \oplus \dots \oplus (1 - 2x_d)$, i.e., the parity, or modulo 2 sum, of $(1 - 2\mathbf{x})$. If \mathbf{x} corresponds to the variable nodes connected to a check node, then the check node constraint is satisfied and $h(\mathbf{x}) = 0$. In the absence of synchronization, however, the components of \mathbf{x} are independent and equiprobable, and $h(\mathbf{x})$ may be either 0 or 1. We may form a log likelihood ratio (LLR) of h , given by

$$\Lambda \triangleq \log \left[\frac{\Pr(h(\mathbf{x}) = 0|\mathbf{y})}{\Pr(h(\mathbf{x}) = 1|\mathbf{y})} \right]. \quad (4)$$

From this, we may express the probability that $h(\mathbf{x}) = 0$, given \mathbf{y} , by manipulating (4) to obtain $\Pr(h(\mathbf{x}) = 0|\mathbf{y}) = 1/(1 + e^\Lambda)$. We may rewrite Λ as

$$\begin{aligned} \Lambda &= \log \left[\frac{f(\mathbf{y}|h(\mathbf{x}) = 0)}{f(\mathbf{y}|h(\mathbf{x}) = 1)} \right] \\ &= \log \left[\frac{\sum_{\mathbf{x}:h(\mathbf{x})=0} f(\mathbf{y}|\mathbf{x})}{\sum_{\mathbf{x}:h(\mathbf{x})=1} f(\mathbf{y}|\mathbf{x})} \right] \\ &= \log \left[\frac{\sum_{\mathbf{x}:h(\mathbf{x})=0} \exp \left[\frac{\langle \mathbf{y}, \mathbf{x} \rangle}{\sigma^2} \right]}{\sum_{\mathbf{x}:h(\mathbf{x})=1} \exp \left[\frac{\langle \mathbf{y}, \mathbf{x} \rangle}{\sigma^2} \right]} \right] \end{aligned} \quad (5)$$

where $\langle \mathbf{y}, \mathbf{x} \rangle = \sum_{i=1}^d y_i x_i$. When $d = 2$, for example, the LLR in (5) becomes

$$\Lambda = \log \left[\frac{\cosh \left(\frac{y_1 + y_2}{\sigma^2} \right)}{\cosh \left(\frac{y_1 - y_2}{\sigma^2} \right)} \right] \quad (6)$$

Approximation to the LLR—When $d > 2$, there are 2^{d-1} exponential terms in the numerator of (5), and 2^{d-1} exponential terms in the denominator. A common approximation to LLR expressions under Gaussian statistics is to replace the sum of exponentials by its largest term. This corresponds to using the nearest neighbor to \mathbf{y} having even parity in the numerator, and the nearest neighbor to \mathbf{y} having odd parity in the denominator:

$$\begin{aligned} \mathbf{x}(0) &\triangleq \operatorname{argmin}_{\mathbf{x} \in \{-1, +1\}^d: h(\mathbf{x})=0} \|\mathbf{y} - \mathbf{x}\| \\ \mathbf{x}(1) &\triangleq \operatorname{argmin}_{\mathbf{x} \in \{-1, +1\}^d: h(\mathbf{x})=1} \|\mathbf{y} - \mathbf{x}\|. \end{aligned}$$

Thus, we may approximate (5) by

$$\Lambda \approx \log \left[\frac{\exp \left[\frac{\langle \mathbf{y}, \mathbf{x}(0) \rangle}{\sigma^2} \right]}{\exp \left[\frac{\langle \mathbf{y}, \mathbf{x}(1) \rangle}{\sigma^2} \right]} \right] \quad (7)$$

It is plain to see that $\mathbf{x}(0)$ and $\mathbf{x}(1)$ differ in exactly one position, namely,

$$m \triangleq \operatorname{argmin}_{m': 1 \leq m' \leq d} |y_{m'}|,$$

¹⁰ Jon Hamkins, "CCSDS attached synchronization marker overhead," JPL Interoffice Memorandum, 332.2008.12.001, December 2008.

and using $x_m(0) = -x_m(1)$, we may rewrite (7) as

$$\Lambda \approx \log \left[\frac{\exp \left[\frac{y_m x_m(0)}{\sigma^2} \right]}{\exp \left[\frac{y_m x_m(1)}{\sigma^2} \right]} \right] \quad (8)$$

$$= \frac{2y_m x_m(0)}{\sigma^2} \quad (9)$$

$$= \lambda_m x_m(0) \quad (10)$$

where $\lambda_m \triangleq 2y_m/\sigma^2$ is an individual channel symbol LLR used in the usual decoding operation.

That is, the LLR of a check node has magnitude dominated by the LLR of the least reliable channel symbol connected to it, and sign governed by whether the *hard decisions* of the channel symbols connecting to the check node satisfy the check node.

The Maximum Likelihood Synchronizer

We now turn to the question, given a contiguous stream of symbols from a binary, linear (n, k) code, which symbol most likely begins a new codeword?

Without an attached sync marker (ASM), the only clue to the correct synchronization offset lies in the structure the code imposes. Any linear code has an associated parity-check matrix, each row of which defines a subset of code symbols that have even parity. Using the analysis above, we may express the maximum likelihood synchronization position as the one that maximizes the LLR in (10), summed over all such (presumed independent) check nodes. The metric at a given position is given by

$$M = \sum_{\text{check nodes } i} \Lambda_i \quad (11)$$

where Λ_i is the LLR of the i th check node, given in (10). This metric is similar to the number-of-satisfied-check-nodes metric, except that it is a soft measure, not a hard one, because of the presence of λ_m in (10).

ACKNOWLEDGMENTS

I thank my JPL colleagues Kevin Quirk for first bringing to my attention the brute force synchronization method for MLC in 2003, Ken Andrews for several conversations about LDPC frame synchronization going back to October 2006, and Sam Dolinar for many helpful recent comments regarding the distributions in Figures 2 and 3.

The research described in this publication was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

[1] TM synchronization and channel coding. CCSDS 131.0-B-1. Blue Book, Issue 1. September 2003,

<http://public.ccsds.org/publications/archive/131x0b1.pdf>.

- [2] Low density parity check codes for use in near-Earth and deep space. CCSDS 131.1-O-2. Orange Book, Issue 2. September 2007, <http://public.ccsds.org/publications/archive/131x1o2e2.pdf>.
- [3] J. I. Statman, B. Siev, and J. Rabkin. Node and frame synchronization in the big Viterbi decoder. *TMO Progress Report*, 42(103):154–160, November 1990, http://ipnpr.jpl.nasa.gov/progress_report/42-103/103P.PDF.

BIOGRAPHY



Jon Hamkins (S'94, M'96, SM'03) received the B.S. degree in electrical engineering from the California Institute of Technology (Caltech), Pasadena, in 1990 and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Illinois at Urbana-Champaign in 1993 and 1996, respectively. Since then, he has been with the Jet Propulsion Laboratory, Caltech, where he is now supervisor of the Information Processing Group. His research interests include error control theory, information theory, autonomous receivers, ranging, and optical communications.