Data Synchronization with Timing*

Navin Kashyap and David L. Neuhoff

Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109
{nkashyap,neuhoff}@eecs.umich.edu

Abstract

This paper proposes and analyzes data synchronization techniques that not only resynchronize after encoded bits are corrupted by insertion, deletion or substitution errors, but also produce estimates of the time indices of the decoded data symbols, in order to determine their positions in the original source sequence. The techniques are based on block codes, and the estimates are of the time indices modulo some integer T, called the *timing span*, which is desired to be large. Several types of block codes that encode binary data are analyzed on the basis of the maximum attainable timing span for a given coding rate R (or, equivalently, redundancy $\rho = 1 - R$) and permissible resynchronization delay D. It is found that relatively simple codes can asymptotically attain the maximum timing span among such block codes, which grows exponentially with delay, with exponent $D(1-R) + o(D)$. Thus large timing span can be attained with little redundancy and only moderate values of delay.

Keywords

cascaded codes, comma-free codes, embedded-index codes, natural marker codes, periodic prefix-synchronized (PPS) codes, prefix-synchronized codes, synchronization delay, sync-timing codes, timing span

*This work was supported by NSF Grants NCR-9415754 and CCR-9815006.
Portions were published in the proceedings of the Data Compression Conference, Snowbird Utah, Mar. 1999, and of the 1999 IEEE Information Theory Workshop, Kruger National Park, South Africa, June 1999.

I. Introduction

To focus sharply on data synchronization issues, consider the situation, illustrated in Figure 1, in which a stream, *i.e.* infinite sequence, of data symbols $x = (x_1, x_2, \ldots)$ from a finite alphabet is encoded into a binary stream $b = (b_1, b_2, \ldots)$ that is connected to a decoder via a switch that closes at some arbitrary finite time. Thus, an arbitrary finite prefix of the encoded bit stream is lost, but no other corruption occurs. We refer to this connection of encoder to decoder as a *switch channel*[1]. The task of the decoder is to produce a data stream $y = (y_1, y_2, \ldots)$ having an infinite suffix, say $(y_n, y_{n+1}, \ldots)$, that is also an infinite suffix of x, say $(x_m, x_{m+1}, \ldots)$. In other words, the decoder should correctly reproduce all but a finite prefix of the original data stream. Since the decoder begins decoding at some arbitrary point in the encoded stream, it will typically produce wrong data for a while before *acquiring synchronization* and correctly reproducing data symbols thereafter. The number of bits received before the acquisition of synchronization is considered to be a resynchronization delay, and the *delay* D of a specific code, *i.e.* an encoder-decoder pair, is defined to be the supremum of the delay over all possible data streams and switch closing times. Clearly, we desire delay to be as small as possible. It may happen that for some data stream and some switch closing time, the decoder never acquires synchronization — for example, if the data alphabet has four symbols which the encoder maps one-at-a-time to binary codewords of length 2. In this case, it is impossible for the decoder to reliably acquire synchronization, since any parsing of the received stream into pairs of bits constitutes a valid encoded stream. In such a situation, the code delay is said to be infinite.

Synchronization codes generally make delay finite by introducing redundancy, which we also desire to be small. In other words, we want the rate of the code to be large, where, as in channel coding, we define *rate* to be the number of data symbols per encoded bit. Generally speaking there is a tradeoff between rate and delay — codes with high rates (low redundancy) tend to have large delays.

So as to focus on data synchronization rather than data compression issues, we consider only codes that are *fixed-rate* in the sense that there are integers i and j such that j bits emerge from the encoder for every i data symbols that enter. Such codes have rate $R = i/j$ data symbols per encoded bit; equivalently, the redundancy of such codes is given by $\rho = (j - i \log_2 A)/j =$

[1]Later, we consider synchronization in the more general setting where the data stream suffers arbitrary insertion, deletion and substitution errors.

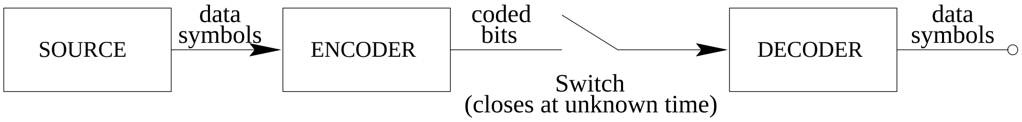

Fig. 1. Block diagram of a communication system needing synchronization

$1 - R\log_2 A$, where A is the cardinality of the data symbol alphabet. We leave the investigation of variable-rate codes to future work.

In many situations, in addition to acquiring data synchronization and reproducing data symbols, it would be quite valuable for the decoder to produce estimates of their time indices, as illustrated in Figure 2. Specifically, the decoder should produce two streams — one of data symbol reproductions $(y_1, y_2, \ldots)$ and the other of integer time indices $(I_1, I_2, \ldots)$. The *time index* of a data symbol is simply its position or ordinal number in the original data stream. Following the usual convention, the symbol with time index i is x_i, the ith symbol produced by the source. When the decoder produces $I_j = i$, then y_j is intended to be the reproduction of x_i.

As a motivating example, suppose the following sequence of temperatures corresponding to a known sequence of cities is encoded for transmission across a switch channel:

$$\begin{array}{rclllllllll} x \text{ (temp in F)} & = & 43 & 64 & 27 & 54 & 36 & 42 & 73 & 45 & 79 & 66\ldots \\ \text{cities} & = & \text{Det} & \text{LA} & \text{Chi} & \text{Bos} & \text{NY} & \text{StL} & \text{Mia} & \text{Bal} & \text{Hou} & \text{Atl}\ldots \\ \text{indices} & = & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10\ldots \end{array}$$

Suppose that due to the loss of a prefix of the encoded bit stream, the decoder produces

$$y \;=\; 73 \quad 40 \quad 54 \quad 36 \quad 42 \quad 73 \quad 45 \quad 79 \quad 66\ldots$$

We see that the decoder has acquired synchronization in time to produce "54" and all subsequent temperatures. However, since the user does not know the cities to which the various temperatures correspond, there is little value in the infinite suffix of correctly decoded data symbols. Instead, we desire the decoder to produce an estimated index to accompany each output symbol, as in

$$\begin{array}{rcccccccccc} y & = & 73 & 40 & 54 & 36 & 42 & 73 & 45 & 79 & 66\ldots \\ I & = & 12 & 7 & 4 & 5 & 6 & 7 & 8 & 9 & 10\ldots \end{array}$$

Codes such as these, which we will call *synchronization with timing codes*, or *sync-timing codes* for short, are the focus of this paper. One possibility is to precede each encoded temperature with an encoding of the city to which it corresponds, but there are better methods. The need for sync-timing codes also arises in video transmission over a noisy channel, or in a scenario where a television viewer is changing channels, causing the decoder input to switch from one data stream to another. In these situations, it is important for the decoder to identify the beginning of each line and frame. Sync-timing codes are also needed in simultaneous audio and video transmission, where it is important that the indices of the decoded audio data match those of the video.

Actually, the sync-timing codes that we explore in this paper have the more limited goal that the decoder produce the time indices modulo some positive integer, T, that we call the *timing span* of the code. For example, if $T = 4$, such a decoder might produce

$$\begin{array}{rcccccccccc} y & = & 73 & 40 & 54 & 36 & 42 & 73 & 45 & 79 & 66\ldots \\ I & = & 4 & 3 & 4 & 1 & 2 & 3 & 4 & 1 & 2\ldots \end{array}$$

The reason for providing only this limited form of timing is that, as shown in Appendix A, there are no sync-timing codes with finite delay and positive rate whose decoder can produce an infinite suffix of the actual time indices, given that an arbitrary finite prefix of the encoded data might be lost. On the other hand, this "modulo" form of timing is not unprecedented. For example, conventional clocks tell time modulo 12 or 24 hours, calendars provide data modulo one year, and many time-stamping cameras and older computers provide dates modulo one century. The key to providing good timing information is simply to choose T wisely. For example, in video transmission, if T corresponds to the length of one line, or a multiple thereof, then the decoder can display lines with the proper horizontal alignment. If T also corresponds to the length of one frame, or a multiple thereof, then the decoder can display lines with the proper vertical alignment, as well. (It will not display the bottom of one frame above the top of the next.) Finally, if T is also large relative to the size of video clips, then the decoder can identify which video clip is being displayed, even if it began decoding at some arbitrary point in the video stream. In simultaneous audio and video transmission, if T corresponds to, say, 1000 seconds, then either the audio and video will be perfectly synchronized, or they will be off by a multiple of 1000 seconds.

Though sometimes, as in the video coding example, T needs to be a multiple of some particular integer, generally speaking, the larger the value of T, the better. To amplify this, note that when the decoder assigns the index value i to a particular decoded data value, the user knows that the actual time index of this data value is some member of the set $\{i, T+i, 2T+i, \ldots\}$. The larger the value of T, the less dense is this set, *i.e.*, the less uncertainty there is about the actual time index. Thus in this paper we will seek codes with large timing span.

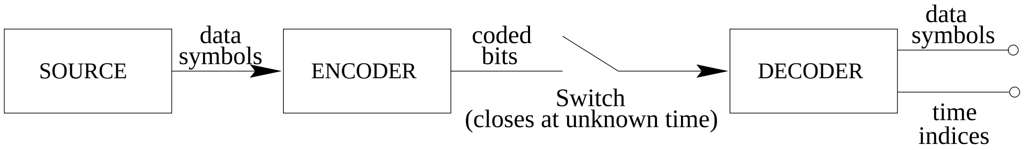

Fig. 2. Block diagram of a communication system needing synchronization with timing information

In summary, the performance of a sync-timing code is characterized by the three quantities: rate R (or redundancy $\rho = 1 - R\log_2 A$), delay D (which is now defined to be the maximum number of received bits until the decoder begins correctly reproducing data symbols and their modulo-T time indices), and timing span T. Having already mentioned the tradeoff between rate R and delay D in conventional synchronization, we now mention that there are, not surprisingly, tradeoffs among all three measures: R, D and T. Principal goals of this paper are to find or bound the region of achievable (R, D, T) performance triples and to find simple codes that come close to achieving these bounds. To characterize this region we will focus mainly on the function $T(r, d)$ giving the maximum timing span of sync-timing codes with rate at least r and delay at most d.

Time indices of the sort produced by sync-timing code are essential in many, if not most, situations where synchronization is needed, and so provision is generally made to embed them in some fashion into the encoded bit stream. For example, the MPEG family of video coding standards embeds time stamps to synchronize the audio and video, and special markers at the beginning of frames and macroblock rows, in order to permit proper alignment of lines and frames. Moreover, to reduce resynchronization delay, it permits the encoder to introduce markers within rows of macroblocks, at the beginning of so-called *slices*, which are horizontally contiguous sets of macroblocks, *cf.* [1], Chap.'s 3, 8, 10. On the other hand, the provision of such timing information is not always done in the communication/coding system; rather it may be left to the user of the system, *e.g.* to the application layer, to provide the necessary encoding, such as sending the city name before the temperature. Nevertheless, valuable communication resources are devoted to such timing information. Therefore, even if it is not always the communications engineer who deals with timing information, it is important for there to be a sound understanding of the fundamentals of synchronization with timing. To our knowledge, this paper is the first work to analyze synchronization codes on the basis of their timing information as well as their rate and delay, and to establish sync-timing codes as a class of codes.

In the rest of this paper, we focus on the important special case of binary data. Though the conventional synchronization problem has a trivial solution in this case, namely, the identity encoder and decoder, yielding a code with zero redundancy and zero delay, there are no such trivial solutions for synchronization with timing span $T > 1$ (note that a sync-timing code with $T = 1$ is just a conventional synchronization code). Indeed, we anticipate that our methodology and results generalize to larger alphabets.

In this paper, we introduce a representative sampling of the large variety of sync-timing codes that can be conceived. This sampling of codes is presented in increasing order of sophistication and performance, beginning in Section II with the class of periodic prefix-synchronized codes, which contains probably the simplest kind of sync-timing codes called natural marker codes, and ending in Section V with the class of comma-free sync-timing codes. *Periodic prefix-synchronized* (PPS) codes, in their simplest form which we call *separable*, map each block of k data symbols into a binary codeword of length $L \geq k$, and precede each codeword with one of p distinct *markers*, $M_1, ..., M_p$, which are binary sequences of length m. The markers and codewords are chosen so the markers are distinctly recognizable in any encoded sequence (they cannot appear within a codeword or in the concatenation of a codeword and a marker) and, consequently, they permit resynchronization after bits are lost, inserted or changed. The sequence with which markers are used is $M_1, \ldots, M_p, M_1, \ldots, M_p$, and so on. This permits a decoder to determine the time index of a decoded data symbol modulo $T = kp$, the timing span of the code. The rate of such a code is $R = k/(L+m)$, and the delay is taken to be $L + m$, the number of bits between markers.

When $p = 1$, PPS codes are just the prefix-synchronized codes first studied by Gilbert [2], who analyzed their rate *vs.* blocklength (*i.e.* delay). Further analysis was undertaken by Guibas and Odlyzko [3], and systematic encoding procedures were developed by Morita *et al* [4]. Comma-free codes, which are a broader class of codes for conventional synchronization, were introduced earlier by Golomb, Gordon, Welch and Delbruck [5] [6], and further studied by Eastman [7], and Golomb and Gordon [8].

Section II also provides a simple upper bound showing that for PPS codes, $T(r, d) \leq d2^{d(1-r)}$. In Sections III and IV, we consider two constructive approaches to PPS code design called *natural marker* and *cascaded* codes, respectively. The former is, arguably, the obvious approach. Roughly speaking, we are able to show that natural marker codes attain $T(r, d) \sim rd2^{d(1-r)/2}$ and cascaded codes attain $T(r, d) \sim 2^{d(1-r)}$. These results show that the maximum achievable timing span of PPS codes grows approximately exponentially with exponent $d(1-r)$, and that simple codes can provide good performance. Moreover, as interesting intermediate results, we obtain bounds on the function $D(p,r)$ giving the least delay of PPS codes with period p and rate r or more.

Section V introduces and studies two other classes of sync-timing codes, referred to as *prefix-synchronized, embedded-index* (PSE) codes and *comma-free sync-timing* (CFST) codes. While these lack the simple structure of separable PPS codes, it is found that their performance is somewhat better than that of cascaded codes. However, their timing span is again bounded by $T(r,d) \leq d2^{d(1-r)}$, so they are not asymptotically better.

The paper concludes in Section VI, with an example and a summary. Some details of Sections I through V are left to appendices.

We conclude the introduction with several remarks. We first point out that although, for simplicity, this section has focused on the switch channel, sync-timing codes are also needed and useful for more realistic channels that make arbitrary insertion, deletion or substitution errors, or some subset thereof. In this case, rate, timing span and delay are still the appropriate measures of performance. The first two are defined as before; delay requires the more general definition that it is the smallest integer d such that whenever a sufficiently large string of bits is received without error, the decoder will acquire data and timing synchronization within at most d bits and, subsequently, decode correctly until it is affected by the next error. For PPS codes, this measure of delay is approximately the same as for the switch channel. Hence, the previously mentioned results on the achievable (R,D,T) performance triples for the switch channel apply also to this more general channel.

We wish to add the cautionary remark that synchronization codes by themselves cannot provide sufficiently good performance in certain situations, for example, when used on a channel that makes arbitrary errors. In such a case, while they limit error propagation to roughly the delay of the code, these codes do tend to increase the frequency of errors, in the sense that even a single channel error may be magnified into a burst of errors whose length is bounded by the delay of the code. Thus, if channel errors occur as frequently as, say, once every D bits, D being the delay of the code, the resulting decoded data is not likely to be very satisfactory. In this situation, it is preferable to first use error-correcting codes to create a channel that makes infrequent errors, over which synchronization codes can offer good performance. Simulation results are available in the literature [9] that demonstrate precisely this point in the case of variable-length codes over the ordinary binary symmetric channel, where again data synchronization is an important issue.

The final remark is to note that while we have defined and discussed the synchronization problem at the data symbol level, in the previous literature, this problem is often considered at a higher level, in the context of *data frames* (cf. [10]). In this context, the encoded bit stream is assumed to be a sequence of binary codewords or frames, each of which encodes some string of data symbols. The synchronization problem then reduces to the problem of finding the correct parsing of the received bits into frames for further decoding. When a frame is decoded into a string of data symbols, the position of each data symbol in that string is known relative to the beginning of the string. Thus, in effect, framing serves to provide timing information within the frame. While the previous work identifies rate and delay as performance measures, it fails to recognize timing span as such. Therefore, neither the tradeoffs nor methods of expanding timing span beyond that obtainable with a simple code such as Gilbert's prefix-synchronized codes have previously been explored.

II. Periodic Prefix-Synchronized Codes

As suggested earlier, we assume that the data source has a binary alphabet, specifically, $\{a, b\}$. *Periodic prefix-synchronized* (PPS) codes are binary block codes that insert synchronizing markers at regular intervals. They are characterized by positive integers p, m, L, non-negative integers $k_1, \ldots, k_p$, distinct binary sequences called markers $M_1, \ldots, M_p$, each of length m, and codebooks $C_1, \ldots, C_p$. Each C_i contains 2^{k_i} binary codewords of length $n = m + L$ from the set $\{0,1\}^n$, with the properties that each codeword begins with the marker M_i, and when a codeword from C_i is followed by one from C_{i+1}, no marker appears at any place except at the beginning of each codeword. Here, C_{p+1} is to be interpreted as C_1.

Associated with each codebook is a bijective encoding rule $e_i : \{a, b\}^{k_i} \to C_i$. Encoding proceeds by applying e_1 to the first k_1 data symbols, e_2 to the next k_2 data symbols, ..., e_p to the next k_p data symbols, e_1 to the next k_1 data symbols, and so on. Overall, one may view the operation as block coding with input and output blocklengths $\mathcal{K} = \sum_{i=1}^{p} k_i$ and $\mathcal{N} = pn$, respectively (see Figure 3). The rate of the code is

$$R = \frac{\mathcal{K}}{\mathcal{N}}$$

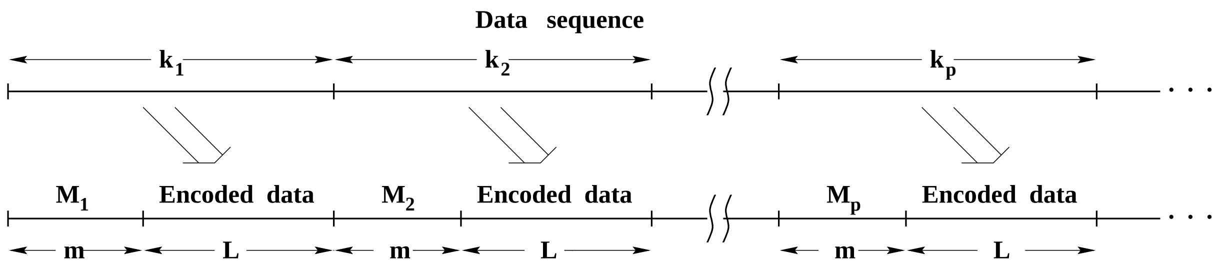

Fig. 3. A fragment of the encoded bit stream for a PPS code.

Since R can never be greater than 1, codes with rates close to 1 are desirable. Equivalently, the redundancy is $\rho = (\mathcal{N} - \mathcal{K})/\mathcal{N} = 1 - R$. A specific PPS code with period p, whose codebooks contain sequences of length n, will be referred to as a (p, n) code, or a (p, m, n) code when m needs to be emphasized. The integer p will be called the *period* of the code.

For $p = 1$, the above description is simply that of a prefix-synchronized code, first studied by Gilbert [2], and subsequently by Guibas and Odlyzko [3] among others.

Example: $p = 2$, $m = 3$, $L = 4$, $k_1 = k_2 = 1$, $M_1 = 111$, $M_2 = 000$, $C_1 = \{1110011, 1110101\}$, and $C_2 = \{0001100, 0001010\}$. Observe that 1110010 could not replace a codeword in C_1 because when it is concatenated with M_2, M_2 appears in a position where it is not permitted — $111001\underline{000}0$.

While there are several potential decoding algorithms, only one will be described here. Let ${e_i}^{-1}$ denote the inverse of e_i. To decode the received binary sequence $y_1, y_2, y_3, \ldots$, the decoder looks for the first occurrence of a marker. If the first marker found is M_i starting at y_{j+1}, for some $i \in \{1, 2, \ldots, p\}$ and some integer $j \geq 0$, then the decoder discards $y_1, \ldots, y_j$. It then checks whether $(y_{j+1}y_{j+2} \ldots y_{j+n}) \in C_i$. If so, the decoder outputs the string of bits $\underline{u} = {e_i}^{-1}(y_{j+1}y_{j+2} \ldots y_{j+n})$, and the string of integers $\underline{I} = (k + 1, k + 2, \ldots, k + k_i)$, where $k = \sum_{j=0}^{i-1} k_j$, defining k_0 to be 0. It then begins another round of decoding, looking for the next marker and checking to see if it is embedded in a valid codeword. Since there can be no marker beginning and ending within $(y_{j+1}y_{j+2} \ldots y_{j+n})$, it suffices to look for markers starting at $y_{j+n-m+2}$. On the other hand, if $(y_{j+1}y_{j+2} \ldots y_{j+n}) \notin C_i$, then the decoder outputs nothing and begins another round of decoding starting at y_{j+2}. It is useful to point out that in the former case, after producing $\underline{u}$ and $\underline{I}$, we could have the decoder begin another round of decoding starting at y_{j+n+1}. While this does not affect the performance of the code when applied to the switch channel, it turns out that in the case of the channel that makes arbitrary insertion, deletion and synchronization errors, the delay due to this modified decoder is roughly twice that due to the decoder described originally.

This decoder, which has a sliding-block structure, as opposed to a block structure, could be used on the switch channel or any channel with insertion, deletion and substitution errors. As required, it outputs a decoded data stream $u_1, u_2, u_3, \ldots$, with each $u_i \in \{a, b\}$, *and* a decoded index stream $I_1, I_2, I_3 \ldots$, with each $I_i \in \{1, 2, \ldots, \mathcal{K}\}$, where $\mathcal{K}$ is the overall input blocklength defined earlier, and where I_i is what the decoder estimates to be the index, modulo $\mathcal{K}$[2], of u_i as it appeared in the data sequence that was input to the encoder. Consequently, the timing span of the code is

$$T = \mathcal{K} = p\, n\, R$$

We now discuss the code's ability to acquire synchronization, and the resulting delay. If the decoder begins operation at some arbitrary point in the sequence of encoded bits and no errors occur, as happens with the switch channel, then the decoder acquires synchronization as soon as it locates a marker. The above algorithm will locate a marker after at most $n-1$ more channel output bits have been discarded, the worst case being when the decoder begins operation at the second bit of a codeword from one of the C_i's.

It is also easy to see that if the channel makes an arbitrary, but finite, number of insertion, deletion and substitution errors, then the decoder begins to decode correctly after discarding or erroneously decoding at most $n-1$ uncorrupted bits beyond the last error. The worst case may occur, for example, when the channel makes a single error, flipping the second bit of the code stream. This may cause the first n bits to no longer form an element of any of the C_i's, but the n bits starting from the second bit may now form a codeword from some C_i. In such a case, the above algorithm erroneously decodes the $n-1$ uncorrupted bits following the error, but acquires synchronization (and timing) at the next marker. By the above reasoning, it follows that if the channel makes bursts of errors that are separated by at least $(1+i)n-1$ uncorrupted bits, for some integer $i > 0$, then the above algorithm correctly decodes at least i codewords after each burst of errors.

It will be our convention that the delay (for acquiring data and timing synchronization) of a (p, n) PPS code with this decoder is

[2] "modulo $\mathcal{K}$" is used here in a sense slightly different from its conventional meaning: we intend $x \pmod{\mathcal{K}}$ to mean $x \pmod{\mathcal{K}} + 1$ in the conventional sense

$$D = n \tag{1}$$

This is in keeping with the fact that the decoder acquires synchronization after discarding or erroneously decoding at most $n-1$ bits beyond an error, in the absence of further errors. We take n to be the definition of delay instead of $n-1$ because it is simpler and still captures the general idea of delay as a measure of the time needed to acquire synchronization and timing. It would also be perfectly reasonable to define delay in terms of the number of data symbols lost between the time of occurrence of an error and the time at which synchronization is recovered, which in this case would be nR. However, for the sake of convenience, we shall use (1) as the definition.

In summary, the performance of a (p, n) PPS code is determined by the triple (R, D, T), with rate R close to 1, small delay D and large timing span T being desirable. Naturally, tradeoffs exist. For example, to increase timing span T, while keeping R fixed, it is necessary to either increase the delay D or period p. The latter may necessitate increasing the marker length m in order to obtain p distinct markers. Even if this is not necessary, an increase in the number of markers imposes additional constraints on the sequences allowable in the C_i's. So if m and n remain fixed, then the k_i's and hence $\mathcal{K}$ may decrease, causing a decrease in the rate. Hence to keep the rate fixed, it may be necessary to increase n, inducing a proportionate increase in $\mathcal{K}$.

As suggested in the introduction, to characterize the set of achievable (R, D, T) triples, which represents a kind of capacity region for PPS codes, we focus on the functions $D(p, r)$, which represents the least delay achievable by any PPS code with period p and rate at least r, $T(p, r, d)$, which gives the maximum timing span achievable by any PPS code with period p, rate at least r and delay at most d, and $T(r, d)$, which gives the maximum timing span achievable by any PPS code with any period, rate at least r and delay at most d. If we define $R(p, n)$ to be the maximum rate achievable by a (p, n) code, if such a code exists, and to be zero otherwise, then we can write

$$D(p, r) = \min_{\{n:\, R(p,n) \geq r\}} n \tag{2}$$

$$T(p, r, d) = \max_{\{n:\, R(p,n) \geq r,\, n \leq d\}} p\, n\, R(p, n) \tag{3}$$

$$T(r, d) = \max_{p} T(p, r, d) \tag{4}$$

If there exists no PPS code with rate at least r, then we define $D(p,r) = \infty$. Similarly, if the constraint sets defining $T(p,r,d)$ or $T(r,d)$ are empty, we define them to be zero.

Clearly, $D(p,r)$ is a non-decreasing function of r, while $T(p,r,d)$ and $T(r,d)$ are non-increasing functions of r. Moreover, $T(p,r,d)$ and $T(r,d)$ are non-decreasing functions of d. The dependence of these functions on p is less clear. However, the tendency is for $D(p,r)$ to increase with p.

Notice that $D(p,r)$ is the smallest value of delay d such that $T(p,r,d) > 0$. Consequently, it can be shown that for $d = D(p,r)$,

$$T(p,r,d) = p\,d\,R(p,d)$$

For $d > D(p,r)$, the value of n achieving the maximum in the definition of $T(p,r,d)$ ordinarily satisfies $n \approx d$ and $R(p,n) \approx r$. Therefore, for $d \geq D(p,r)$,

$$T(p,r,d) \approx p\,d\,r$$

which indicates the potential benefits of increasing p, namely, that timing span increases approximately in proportion to p, provided d is large enough that $T(p,r,d)$ is not zero[3]. The key question then becomes: How rapidly does $D(p,r)$ increase with p? If it increases slowly, then at a given delay d, p can be chosen large and the timing span will be large.

The following result provides a lower bound to $D(p,r)$ and an upper bound to $T(r,d)$.

Theorem 1: (i) For $r \in (0,1)$ and $p > 0$,

$$D(p,r) \geq \frac{\log_2 p}{1-r} \tag{5}$$

(ii) For $r \in (0,1)$ and $d \geq 0$,

$$T(r,d) \leq d\,2^{d(1-r)} \tag{6}$$

Proof: (i) Recall that $D(p,r)$ is the least delay among all PPS codes with period p and rate at least r. We show that any (p,n) PPS code with rate $R \geq r$ has delay $D = n \geq (\log_2 p)/(1-r)$.

The idea is that having markers of length m in a code of rate R places a lower limit on the length n of each codeword. Since the information content of a codeword is nR bits, the redundancy in the codeword is $n(1-R)$ bits. Since the marker is part of the redundancy, we have $m \leq n(1-R)$, and hence

$$n \geq \frac{m}{1-R} \geq \frac{\log_2 p}{1-r}$$

[3]In fact, in Appendix C, we show that for all sufficiently large d, $T(p,r,d) \geq pdr$.

the last inequality following from the fact that there are p distinct markers of length m each, so that we must have $m \geq \log_2 p$.

(ii) Since $T(r,d)$ is the maximum timing span that can be provided by a code with rate at least r and delay at most d, it is sufficient to show that any (p,n) code with rate $R \geq r$ and delay $D \leq d$ has timing span $T \leq d\, 2^{d(1-r)}$.

From the proof of the first part of this theorem, we know that any (p,n) code with rate $R \geq r$ has delay $D = n \geq (\log_2 p)/(1-r)$, so that $p \leq 2^{D(1-r)}$. Hence, for a code with $R \geq r$ and $D \leq d$,

$$T = p\, DR \leq 2^{D(1-r)}\, DR \leq 2^{d(1-r)}\, d$$

where we have used the fact that $R \leq 1$. ■

Obviously, various generalizations of PPS codes are possible. For example, we could allow the code alphabet to be non-binary. We could remove the requirement that all markers have the same length. Or, we might even allow each codebook C_i to contain sequences of length n_i, without the restriction that all n_i's be equal. However, in this paper, we shall only consider PPS codes as described above. Indeed, in Sections III and IV, we shall focus on PPS codes with the additional constraint of *separability*. Specifically, for a separable (p,n) PPS code there is a base codebook C_o containing codewords of length $n-m$ such that each C_i consists of the codewords in C_o, preceded with the ith marker M_i. The markers and the base codebook are chosen so that no marker appears in the concatenation of a marker, a codeword and another marker, except at the beginning and end. In a companion paper [11], we present an exact analysis of optimal non-separable PPS codes for $p=2$. Here, the non-separability simplifies the analysis, because each codebook is designed knowing the marker with which it begins and the marker with which it is followed, whereas for separable codes, the base codebook is designed so that the base codewords could be preceded and followed by arbitrary markers.

III. Natural Marker Codes

As a first PPS code construction, we introduce *natural marker codes*, which are probably the most elementary of sync-timing codes. They provide useful benchmarks and bounds to optimal performance. It is also interesting to learn how well such conceptually simple codes can perform. A (p,n) natural marker code is a separable (p,n) PPS code whose ith marker, M_i, consists of a *flag*, followed by a zero, followed by a binary representation, in $\lceil \log_2 p \rceil$ bits, of the index i. As the indices to be represented are $\{1, 2, \ldots, p\}$, index i is denoted by the usual binary representation of $i - 1$. The flag is the sequence 1^{m_1}, *i.e.*, m_1 ones, where $m_1 = \lceil \log_2 p \rceil + 1$. Thus, the length of each marker is $m = 2\lceil \log_2 p \rceil + 2$. The base codebook C_o contains codewords of length $L = n - m$ that begin and end with a zero and do not contain the flag, *i.e.*, do not contain m_1 consecutive ones. Let $g(m_1, \mathcal{L})$ denote the number of binary sequences of length $\mathcal{L}$ that do not contain m_1 consecutive ones. Thus, $g(m_1, L-2)$ is the number of sequences of length L that are valid candidates for inclusion in the base codebook. The base codebook is chosen to consist of $2^{\lfloor \log_2 g(m_1, L-2) \rfloor}$ such sequences. One may easily check that with this choice of markers and base codebook, markers will not appear within a codeword nor in the concatenation of a codeword and a marker. Thus, natural marker codes are valid separable (p, n) PPS codes.

The essential idea of a natural marker code is that the flag is a recognizable sequence, appearing only at the beginning of each marker, that can be used for synchronization. The index i provides timing information. As an example, for $p = 4$, the set of natural markers is $\{111000, 111001, 111010, 111011\}$. Incidentally, the MPEG video coding standard uses similar markers. For example, before the encoding of each slice (a horizontally contiguous set of macroblocks), there is a marker whose first component is a flag consisting of 23 or more zero's followed by a one, and whose second component is, in effect, a pair of integers indicating the vertical and horizontal position of the slice.

We now comment on the various choices made in defining natural marker codes. The zeros after the flag and at the beginning and end of each base codeword prevent the flag from appearing in unintended places. The all-ones flag is chosen because it is conceptually simple (*e.g.* searching for flags is easy, and inserting the aforementioned zeros prevents the spurious occurrence of flags) and because Gilbert [2] showed that for any m_1, the all-ones marker is optimal for prefix-synchronized codes with markers of length m_1 and all sufficiently large codelengths. From this, it can be shown that choosing the all-ones flag permits the base codebook C_o to be, approximately, as large as possible. Moreover, there exists a simple and efficient algorithm [12] for generating the set of all sequences of length L that do not contain m_1 consecutive ones, which results in efficient encoding and decoding procedures.

Actually, one could consider $1^{m_1}0$ to be the flag, and choose the base codebook to be a set of sequences of length $L = n - m$ that begin with a zero and do not contain $1^{m_1}0$. The weaker constraints on the base codebook (the codewords no longer need to end with a zero, and the "no $1^{m_1}0$" constraint is weaker than the "no 1^{m_1}" constraint) allow an increase in its cardinality. As described in Appendix D, this permits somewhat better performance — roughly doubling the timing span at a given rate and delay — than that obtained with natural marker codes. An efficient algorithm for encoding with this base codebook is described in [4].

The length of the flag, m_1, is chosen to be $\lceil \log_2 p \rceil + 1$ so that it is as short as possible, while not allowing the flag to be contained in the binary representation of any marker index. However, we note that when p is not a power of two, $1^{\lceil \log_2 p \rceil}$ need not be used to represent any index. As a result, one could take m_1 to be $\lceil \log_2 p \rceil$, thus reducing the length of the marker, while still satisfying the constraint that 1^{m_1} should not appear in any marker index. Nevertheless, since the resultant gain is marginal, and so as to avoid having to deal separately with the case when p is a power of two, we shall continue to take m_1 to be as defined previously.

On a related note, it is not always true that the shortest possible flags are the best — an increase in the length of the flag reduces the constraints on the base codebook, which sometimes permits a more than compensating reduction in the length of the base codewords needed to attain a certain rate. When m_1 is permitted to be larger than $\lceil \log_2 p \rceil + 1$, the resulting codes will be called *generalized natural marker codes.*

The performance of natural marker codes is intimately related to the functions $g(m_1, \mathcal{L})$ and

$$\widehat{R}(m_1, \mathcal{L}) = \frac{\lfloor \log_2 g(m_1, \mathcal{L}) \rfloor}{\mathcal{L}} \tag{7}$$

which represents the rate of a code whose codebook consists of sequences of length $\mathcal{L}$ that do not contain m_1 ones. Specifically, the rate of the (p, n) natural marker code is

$$\begin{aligned} R_N(p,n) &= \widehat{R}(m_1, n - 2m_1 - 2)\ \frac{n - 2m_1 - 2}{n} \\ &= \widehat{R}(\lceil \log_2 p \rceil + 1, n - 2\lceil \log_2 p \rceil - 4)\ \frac{n - 2\lceil \log_2 p \rceil - 4}{n} \end{aligned} \tag{8}$$

From $R_N(p,n)$, one can straightforwardly determine $D_N(p,r)$, $T_N(p,r,d)$, and $T_N(r,d)$, which are defined as in (2), (3) and (4), but with the constraint sets restricted to natural marker codes.

When $\mathcal{L} < m_1$, $g(m_1, \mathcal{L}) = 2^{\mathcal{L}}$ since the sequences are too short to contain 1^{m_1}. For $\mathcal{L} = m_1$, $g(m_1, \mathcal{L}) = 2^{m_1} - 1$, since the only sequence of length m_1 that contains 1^{m_1} is 1^{m_1} itself. Similarly, $g(m_1, m_1 + 1) = 2^{m_1+1} - 3$. More generally, there exist recursions (see Appendix B) which one can use to compute $g(m_1, \mathcal{L})$ from the values of $g(m_1, \mathcal{L}')$, $\mathcal{L}' \leq \mathcal{L}$. A particularly important fact, seen

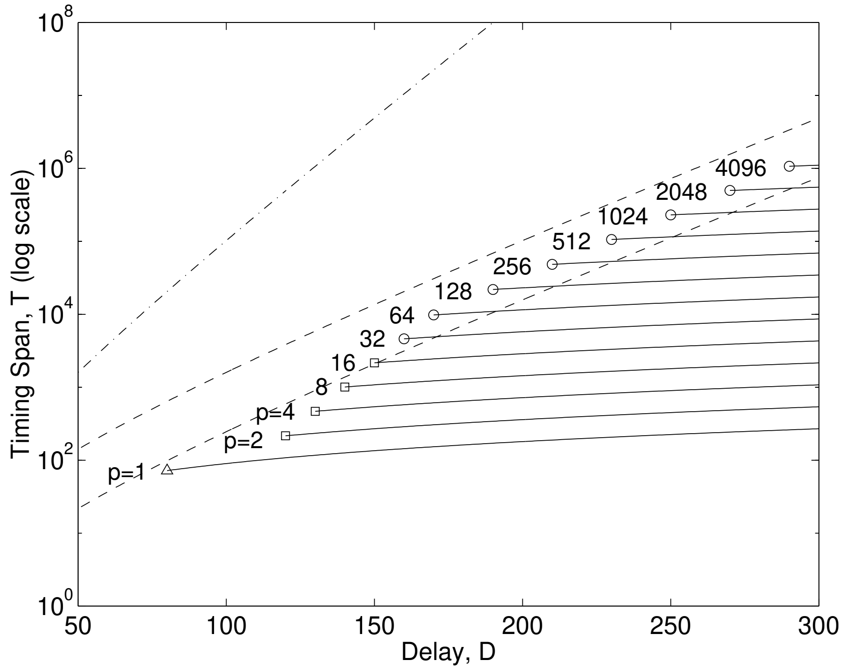

Fig. 4. Timing span vs. delay for natural marker codes. The circles plot $T_N(p,r,d)$ for $r = 0.9$, $d = D_N(p,r)$; the solid lines show the approximations $T_N(p,r,d) \approx dpr$; the dashed line at the top is the upper bound to $T(r,d)$, while the two at the bottom show upper and lower bounds to $T_N(r,d)$.

from Lemma B.2, is that $\lfloor \log_2 g(m_1, \mathcal{L}) \rfloor \geq \mathcal{L} - 1$, provided $\mathcal{L}$ is not too large ($\mathcal{L} \leq 2_1^m + m_1 - 2$).

The aforementioned recursions were used to compute $R_N(p,n)$, and Figure 4 plots, as circles, the resulting values of $T_N(p,r,d)$ for $r = 0.9$ (10% redundancy), p equal to a power of 2 greater than 16, and $d = D_N(p,r)$, which is the smallest value of d for which $T_N(p,r,d) > 0$. For $d > D_N(p,r)$, we plot the approximation $T_N(p,r,d) \approx p\,d\,r$ as solid lines extending to the right from the circles[4]. These lines illustrate that the timing span achievable by a code with period p_1 is less than that achievable by a code with period $p_2 > p_1$, by a factor proportional to p_2/p_1, provided d is large enough that $T_N(p_2,r,d)$ is not zero. We have omitted points for p not a power of 2 because, in such cases, $R_N(p,n)$, and consequently, $D_N(p,r)$, are the same as those for $p' = 2^{\lceil \log_2 p \rceil}$. However, with $d = D_N(p,r)$, $T_N(p,r,d) = p\,R_N(p,d)\,d$ is strictly less than that for p'. For $p \leq 16$, it turns out that $D_N(p, 0.9) = \infty$, *i.e.* there are no period p natural marker codes with rate at least 0.9. However, as shown in Appendix C, there do exist generalized natural marker codes for any given p and $r < 1$. Accordingly, for $p = 2, 4, 8, 16$, the squares plotted in

[4]Proposition C.1 of the appendix could be sharpened to show that for all sufficiently large p, $T_N(p,r,d) \geq p\,d\,r$ for all sufficiently large d.

Figure 4 represent the performance of the best generalized natural marker codes. Also plotted is a triangle representing the performance of the best prefix synchronized codes ($p = 1$) with rate at least r, as determined from the results of Gilbert [2]. Finally, the dashed lines illustrate the upper bound in Theorem 1, as well as the upper and lower bounds derived specifically for natural marker codes in Theorem 4 below. It can be seen that the timing span of natural marker codes seems to grow exponentially with delay, but not as fast as the exponential rate of growth of the upper bound in Theorem 1.

We now derive bounds to $D_N(p,r)$ from which bounds to $T_N(r,d)$ will subsequently be found.

Theorem 2: (i) For $p \geq 2$ and $r \in (0,1)$,

$$D_N(p,r) \geq \frac{2\log_2 p + 2}{1-r}$$

(ii) For each $r \in (0,1)$, there exists an integer $p(r)$ such that for all $p > p(r)$,

$$D_N(p,r) \leq \left\lceil \frac{2\lceil \log_2 p \rceil + 5}{1-r} \right\rceil$$

Proof: (i) We shall show that any (p,n) natural marker code with rate $R \geq r$ has delay $D = n \geq 2(\log_2 p + 1)/(1-r)$. As shown in the proof of Theorem 1, any (p,m,n) PPS code with rate $R \geq r$ must have $n \geq m/(1-R)$. Since a (p,n) natural marker code has markers of length $m = 2\lceil \log_2 p \rceil + 2$, for any such code with rate $R \geq r$, we have

$$n \geq \frac{m}{1-R} \geq \frac{2\log_2 p + 2}{1-r}$$

which proves the first part of the theorem.

(ii) It is sufficient to show that for all sufficiently large p, a (p,n) natural marker code with $n = \lceil (2\lceil \log_2 p \rceil + 5)/(1-r) \rceil$ has rate $R_N(p,n) \geq r$. As usual, the flag has length $m_1 = \lceil \log_2 p \rceil + 1$, the marker has length $m = 2m_1$, and the codewords have length $L = n - m$. From (8) we know $R_N(p,n) = \widehat{R}(m_1, L-2)\,(L-2)/n$, and Lemma B.2 in Appendix B shows

$$\widehat{R}(m_1, L-2) \geq \frac{L-3}{L-2} \tag{9}$$

provided $L - 2 \leq 2^{m_1} + m_1 - 2$. To check this, we note that by the choice of n, m_1 and L,

$$L = n - 2m_1 = \left\lceil \frac{2m_1 + 3}{1-r} \right\rceil - 2m_1$$

from which it is easy to see that $L - 2 \leq 2^{m_1} + m_1 - 2$ for all sufficiently large m_1, or equivalently, for all sufficiently large p, say $p > p(r)$. Therefore, (9) and (8) imply

$$R_N(p,n) \geq \frac{L-3}{n} = \frac{n-m-3}{n} \geq r$$

where the last inequality follows from the fact that $(n-m-3)/n \geq r$ if and only if $n \geq (m+3)/(1-r)$, which is satisfied by $n = \lceil (m+3)/(1-r) \rceil$. This completes the proof of (ii). ■

As mentioned earlier, when p is not a power of 2, it is possible to reduce the length of the natural markers by 1 bit. If these reduced length markers were used to generate natural marker codes, then the above upper bound can be sharpened slightly for all sufficiently large p that are not powers of 2. For all such p, it can then be shown that $D_N(p,r) \leq \lceil (2\lceil \log_2 p \rceil + 4)/(1-r) \rceil$. The proof of this is along the lines of the one above.

As an immediate consequence of the preceding theorem, we have the following corollary.

Corollary 3: For $r \in (0,1)$

$$\lim_{p \to \infty} \frac{D_N(p,r)}{\log_2 p} = \frac{2}{1-r}$$

This result shows that for large p, the best natural marker codes with period p and rate at least r have codelength $n \approx (2\log_2 p)/(1-r) \approx m/(1-r)$. In other words, the length of the markers in the best possible such codes is approximately $n(1-r)$, *i.e.*, nearly all the redundancy in the code lies in the markers, which supports the earlier claim that the all 1's flag leads to large base codebooks. However, note that the codelength is twice the lower bound derived in Theorem 1 for the best possible PPS codes with period p and rate at least r.

We now use the bounds just derived for $D_N(p,r)$ to determine bounds to $T_N(r,d)$, which represents the maximum timing span of natural marker codes with rate at least r and delay at most d. It can straightforwardly be shown that $T_N(r,d)$ is a non-increasing function of r, and a non-decreasing function of d.

Theorem 4: (i) For $r \in (0,1)$ and $d \geq 0$,

$$T_N(r,d) \leq d\, 2^{\frac{d(1-r)}{2} - 1}$$

(ii) For each $r \in (0,1)$, there exists an integer $d(r)$ such that for all $d > d(r)$,

$$T_N(r,d) \geq r\, d\, 2^{\frac{(d-1)(1-r)-7}{2}} (1 + o_d(1))$$

where $o_d(1)$ is a correction term that goes to 0 as d goes to ∞.

These are the bounds for natural marker codes plotted in Figure 4. Note that the plot for the lower bound ignores the $o_d(1)$ correction factor. Note also that it can be seen from the proofs that the bounds in Theorems 2 and 4 apply, as well, to generalized natural marker codes (in the sense of Appendix C).

Proof: (i) It suffices to show that any (p, n) natural marker code with rate $R \geq r$ and delay $D = n \leq d$ has timing span $T \leq d\, 2^{d(1-r)/2-1}$. Accordingly, assume p and n are so chosen. Then as shown in the proof of Theorem 2 part (i), $n \geq (2\log_2 p + 2)/(1-r)$, which upon solving for p yields

$$p \leq 2^{\frac{n(1-r)}{2}-1} \leq 2^{\frac{d(1-r)}{2}-1}$$

Now using the fact that $R \leq 1$, we have

$$T = p\, D\, R \leq 2^{\frac{d(1-r)}{2}-1}\, d$$

which completes the proof of (i).

(ii) Let p be any integer such that $D_N(p, r) \leq d$. By the definition of $D_N(p, r)$, there exists a (p, n) natural marker code with delay $D = D_N(p, r)$ and rate $R \geq r$. The timing span of this code is at least $p\, D_N(p, r)\, r$. Taking the maximum timing span over all such p, we see that

$$\begin{aligned} T_N(r, d) &\geq \max_{\{p:\ D_N(p,r) \leq d\}} p\, D_N(p, r)\, r \qquad (10) \\ &\geq \max_{\{p:\ D_N(p,r) \leq d\}} p\, \frac{2\log_2 p + 2}{1-r}\, r \qquad \text{(from Theorem 2)} \\ &\geq \max_{\{p:\ p > p(r),\ D_N(p,r) \leq d\}} p\, \frac{2\log_2 p + 2}{1-r}\, r \end{aligned}$$

$p(r)$ being the smallest integer such that for all $p > p(r)$, $D_N(p, r) \leq \lceil (2\lceil \log_2 p \rceil + 5)/(1-r) \rceil$, the upper bound derived in Theorem 2. Since this upper bound is at most $(2\log_2 p + 7)/(1-r) + 1$, we can continue the above chain of inequalities as follows:

$$\begin{aligned} T_N(r, d) &\geq \max_{\{p:\ p > p(r),\ (2\log_2 p + 7)/(1-r) + 1 \leq d\}} p\, \frac{2\log_2 p + 2}{1-r}\, r \\ &= \max_{\{p:\ p > p(r),\ p \leq 2^{(d-1)(1-r)/2}\, 2^{-7/2}\}} p\, \frac{2\log_2 p + 2}{1-r}\, r \end{aligned}$$

Now, there exists an integer $d(r)$ such that for all $d > d(r)$, $\lfloor 2^{(d-1)(1-r)/2}\, 2^{-7/2} \rfloor > p(r)$, therefore, the constraint set in the above maximum is not empty. Since the objective function increases monotonically with p, the maximum is attained by $p_0 = \lfloor 2^{(d-1)(1-r)/2}\, 2^{-7/2} \rfloor$, from which we conclude that for $d > d(r)$,

$$T_N(r,d) \geq p_0\, r\, \frac{2\log_2 p_0 + 2}{1-r}$$

To simplify this result, we ignore the floor function in the expression for p_0, and introduce a correction term in the expression that results on substituting for p_0 in the above bound. Thus,

$$\begin{aligned} T_N(r,d) &\geq r\, 2^{(d-1)(1-r)/2}\, 2^{-7/2}\, \frac{(d-1)(1-r)-5}{1-r}\, (1+o_d(1)) \\ &= r\, 2^{(d-1)(1-r)/2}\, 2^{-7/2} \left(d - 1 - \frac{5}{1-r}\right) (1+o_d(1)) \\ &= r\, 2^{(d-1)(1-r)/2}\, 2^{-7/2}\, d\, (1+o_d(1))\, (1+o_d(1)) \end{aligned}$$

which, after merging the terms containing the o-notation, proves the theorem. ■

Corollary 5: For $r \in (0,1)$,

$$\lim_{d\to\infty} \frac{\log_2 T_N(r,d)}{d} = \frac{1-r}{2}$$

Thus, we have shown that for any $r \in (0,1)$ and large enough d, $T_N(r,d)$ increases exponentially with d, with the rate of exponential increase being $(1-r)/2$. While this rate of exponential growth is only half that suggested by the upper bound for $T(r,d)$ in Theorem 1, it is nevertheless interesting to know that the timing span of PPS codes can grow exponentially with delay. It also follows from the above derivation that natural marker codes operating near $T_N(r,d)$ have $p \approx 2^{d(1-r)/2}$, so that their marker length m is approximately $2\log_2 p \approx d(1-r)$. This is consistent with what was deduced from the corollary to Theorem 2, since the delay of a code is equal to the length of its codewords.

IV. Cascaded codes

As shown in the previous section, the redundancy of natural marker codes is dominated by the length of the markers. Therefore, it makes sense to try to reduce this length. Clearly, one cannot hope to reduce the number of bits needed to describe the marker index below $\log_2 p$. However, in this section, we show that the flag length can be reduced, with substantial benefit.

The constraint that governs the design of flag-followed-by-index type markers is that the flag should not appear within the representation of the index of any marker, nor in the concatenation of the flag and the index. Natural marker codes insure this simply by making the flag longer than the binary representation of the index, *i.e.* $m_1 > \lceil \log_2 p \rceil$, and by putting a zero between the flag and the index.

A better approach is to allow the flag to have length $m_1 < \lceil \log_2 p \rceil$, but to require that the marker indices be represented by sequences of some length m_2 that do not contain the flag. We continue to choose the flag to be the all-ones sequence of length m_1 because, as discussed in the previous section, this choice of flag tends to maximize the size of the base codebook. Clearly, m_1 and m_2 must be chosen so that the number of sequences of length m_2 that do not contain the flag 1^{m_1} is at least p, *i.e.* so that $g(m_1, m_2) \geq p$. This will generally require m_2 to be somewhat larger than $\lceil \log_2 p \rceil$, the number of bits used by natural marker codes. The reduction in m_1 will also decrease $g(m_1, n)$, which is the maximum size of the base codebook. However, a small increase in m_2 and the base codebook redundancy will be more than offset by the large decrease in marker length.

PPS codes of the type sketched above will be called *cascaded codes*, because they cascade a flag, a code for the marker index and a code for the data. Specifically, a (p, m_1, m_2, n) cascaded code is a separable (p, n) PPS code whose ith marker, M_i, consists of the flag 1^{m_1}, followed by a zero, followed by the index of i represented as a sequence of length m_2 that does not contain the flag. (The precise representation of the indices does not matter.) Thus, the length of each marker is $m = m_1 + m_2 + 1$. The base codebook is the same as for natural marker codes, which means that it consists of $2^{\lfloor \log_2 g(m_1, n-m-2) \rfloor}$ sequences of length $n - m$ that begin and end with zero and do not contain the flag. The rate of a (p, m_1, m_2, n) cascaded code is

$$R_C(p, m_1, m_2, n) = \widehat{R}(m_1, L) \times \frac{L}{n} \tag{11}$$

where $L = n - m_1 - m_2 - 3$ and, as before, $\widehat{R}(m_1, L) = \lfloor \log_2 g(m_1, L) \rfloor / L$. Let $R_C(p, n)$ denote the maximum rate of (p, m_1, m_2, n) cascaded codes.

For a given r and p, one may find $D_C(p, r)$, which is the least delay of cascaded codes with period p and rate at least r, by finding the smallest n for which there is m_1 and m_2 such that $g(m_1, m_2) \geq p$ and $g(m_1, n - m_1 - m_2 - 3) \geq 2^{nr}$. While this could be done by starting with small values of n, and checking all m_1, m_2 such that $m_1 + m_2 < n$, it typically suffices to check all m_1 up to approximately $2 \log_2 \log_2 p$, and all m_2 up to approximately $2 \log_2 p$. For $r = 0.9$ and various values of p, Figure 5 plots, as circles, points (d, t) where $d = D_C(p, r)$ and

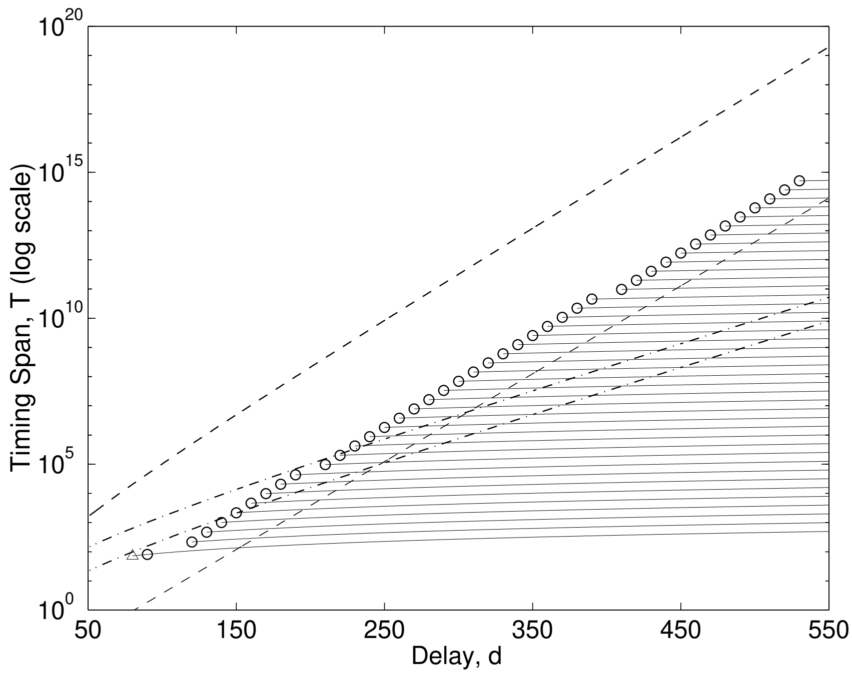

Fig. 5. Timing span vs. delay for cascaded codes with rate at least 0.9. The circles plot $T_C(p,r,d)$ for $r = 0.9$ and $d = D_C(p,r)$; the solid lines show the approximations $T_C(p,r,d) \approx dpr$; the two dashed lines at the top are the upper bound to $T(r,d)$ and the lower bound to $T_C(r,d)$; and the two lowest dashed lines are the upper and lower bounds to $T_N(r,d)$, the timing span of natural marker codes.

$t = p\,d\,R_C(p,d) = T_C(p,r,d)$. The latter function is the maximum timing span of cascaded codes with period p, rate at least r, and delay at most d. For $d < D_C(p,r)$, $T_C(p,r,d) = 0$, and for $d > D_C(p,r)$, the approximation $T_C(p,r,d) \approx p\,d\,r$ is illustrated in the form of solid lines extending to the right of the circles. Also shown, with dashed lines, are the upper and lower bounds for natural marker codes given in Theorem 4, the upper bound from Theorem 1 which applies to all PPS codes, and a lower bound derived below specifically for cascaded codes. One can see that for $d > 200$, cascaded codes have much better performance than natural marker codes. Though for smaller values of d it is difficult to judge from the plot, it turns out that the cascaded codes outperform the natural marker codes at all delays.

The proof of the next theorem, which develops an upper bound on $D_C(p,r)$, shows how m_1 and m_2 may be chosen to good effect when p is large.

Theorem 6: For each $r \in (0,1)$, there exists an integer $p(r)$ such that for all $p > p(r)$,

$$D_C(p,r) \leq \frac{1}{1-r}\,(\log_2 p + \log_2 \log_2 p) + \gamma(r)$$

where

$$\gamma(r) = \begin{cases} \frac{1}{1-r}\,(\frac{r}{1-r}+7)+1, & \text{if } r < 1/3 \\ \frac{1}{1-r}\,(\log_2 \frac{r}{1-r}+8)+1, & \text{if } r \geq 1/3 \end{cases}$$

Proof: We prove this theorem by demonstrating that, given an $r \in (0,1)$, for large enough p, there exists a (p, m_1, m_2, n) cascaded code with rate $R_C(p, m_1, m_2, n) \geq r$, where

$$\begin{aligned} m_1 &= \lceil \log_2 \log_2 p + \beta(r) \rceil \\ m_2 &= \lceil \log_2 p \rceil + 1 \\ n &= \left\lceil \frac{m_1 + m_2 + 4}{1-r} \right\rceil \end{aligned}$$

$\beta(r)$ being $r/(1-r)$ for $r < 1/3$, and $\log_2 \frac{r}{1-r} + 1$ for $r \geq 1/3$. Note that $\beta(r) \geq 0$ for all r.

We first show that there exists a (p, m_1, m_2, n) cascaded code by using Lemma B.2 in the Appendix B with $L = m_2$ to verify that $g(m_1, m_2) \geq p$. Observe that

$$\begin{aligned} 2^{m_1} + m_1 - 2 &\geq 2^{\log_2 \log_2 p + \beta(r)} + \log_2 \log_2 p + \beta(r) - 2 \\ &= 2^{\beta(r)} \log_2 p + \log_2 \log_2 p + \beta(r) - 2 && (12) \\ &\geq \lceil \log_2 p \rceil + 1 \;=\; m_2 && (13) \end{aligned}$$

with the last inequality being true for large enough p, say $p \geq p_1(r)$. Therefore, Lemma B.2 shows that for $p \geq p_1$,

$$g(m_1, m_2) \geq 2^{m_2 - 1} \geq p$$

where the last inequality follows from the choice of m_2.

We next show that the rate of a (p, m_1, m_2, n) cascaded code is at least r. Again we apply Lemma B.2 — this time with $L = n - m_1 - m_2 - 3$. We need to verify that $L \leq 2^{m_1} + m_1 - 2$. From our choices of n, m_1 and m_2, we have

$$\begin{aligned} L &\leq \frac{m_1 + m_2 + 4}{1-r} + 1 - m_1 - m_2 - 3 = (m_1 + m_2)\frac{r}{1-r} + \frac{2+2r}{1-r} \\ &= (\lceil \log_2 \log_2 p + \beta(r) \rceil + \lceil \log_2 p \rceil + 1)\,\frac{r}{1-r} + \frac{2+2r}{1-r} \\ &= \frac{r}{1-r} \log_2 p + o(\log_2 p) && (14) \end{aligned}$$

where $o(\log_2 p)$ denotes a quantity that becomes small relative to $\log_2 p$ as p increases. On the other hand, from (12) and the fact that $2^{\beta(r)} \geq 2r/(1-r)$, we have

$$\begin{aligned} 2^{m_1} + m_1 - 2 &\geq 2^{\beta(r)} \log_2 p + \log_2 \log_2 p + \beta(r) - 2 \\ &\geq 2\frac{r}{1-r} \log_2 p + o(\log_2 p) \end{aligned} \tag{15}$$

From (14) and (15), we see that for all sufficiently large p, say $p \geq p_2(r)$, $L \leq 2^{m_1} + m_1 - 2$. Therefore, Lemma B.2 shows that for such p, $\widehat{R}(m_1, L) \geq (L-1)/L$. From (11) and our choice of n, we find

$$\begin{aligned} R_C(p, m_1, m_2, n) &= \frac{L-1}{n} = 1 - \frac{m_1 + m_2 + 4}{n} \\ &\geq 1 - \frac{m_1 + m_2 + 4}{(m_1 + m_2 + 4)/(1-r)} = r \ , \ p \geq p_2 \end{aligned}$$

Hence, for $p \geq p(r) = \max(p_1(r), p_2(r))$, a (p, m_1, m, n) cascaded code has rate at least r. Finally, the delay of this code is $n \leq \frac{1}{1-r} \left(\log_2 p + \log_2 \log_2 p + \beta(r) + 7\right) + 1$. ■

Theorems 1 and 6 lead immediately to the following corollary.

Corollary 7: For $r \in (0,1)$,

$$\lim_{p \to \infty} \frac{D_C(p,r)}{\log_2 p} = \frac{1}{1-r}$$

Thus, we see that it is possible for PPS codes, specifically cascaded codes, to asymptotically achieve the lower bound on $D(p,r)$ that was derived in Section II. Comparing this result with the corresponding result for natural marker codes, we see that for large p, there exist period-p cascaded codes with rate at least r that have delay, or equivalently codelength, about half that of the best natural marker codes with period p and rate at least r. This gain is principally due to reducing the flag length from $\lceil \log_2 p \rceil + 1$ to approximately $\log_2 \log_2 p$, while increasing the bits required to describe the markers from $\lceil \log_2 p \rceil$ to just $\lceil \log_2 p \rceil + 1$. Thus, the total marker length has been cut approximately in half. Note that like well-designed natural marker codes, well-designed cascaded codes have marker length, $m \approx n(1-r)$, which shows that almost all the redundancy in the code lies in the markers. Equivalently, such codes have $n \approx m/(1-r)$, which shows why their delay is, approximately, proportional to their marker length.

The next result provides an asymptotic lower bound to $T_C(r,d)$, the maximum timing span of cascaded codes with rate at least r and delay at most d.

Theorem 8: For each $r \in (0,1)$, there exists an integer $d(r)$ such that for all $d > d(r)$,

$$\begin{aligned} T_C(r,d) &\geq 2^{(d-\gamma(r))(1-r)+\log_2 \frac{r}{1-r}} - r(d-\gamma(r)) - \frac{r}{1-r}\log_2 e \\ &= 2^{(d-\gamma(r))(1-r)+\log_2 \frac{r}{1-r}}\,(1+o_d(1)) \end{aligned}$$

where $\gamma(r)$ is as defined in Theorem 6. In particular, for $r \geq 1/3$,

$$T_C(r,d) \geq 2^{(d-1)(1-r)-8}\,(1+o_d(1))$$

Proof: Corresponding to (10) in the proof of Theorem 4, we have

$$\begin{aligned} T_C(r,d) &\geq \max_{\{p:\, D_C(p,r)\leq d\}} p\, D_C(p,r)\, r \\ &\geq \max_{\{p:\, D_C(p,r)\leq d\}} p\, \frac{\log_2 p}{1-r}\, r \end{aligned}$$

the second inequality following from (5), since $D_C(p,r) \geq D(p,r)$. Now, by Theorem 6, we may choose $p(r)$ to be an integer such that for all $p \geq p(r)$, $D_C(p,r) \leq (\log_2 p + \log_2\log_2 p)/(1-r) + \gamma(r)$. Then,

$$\begin{aligned} T_C(r,d) &\geq \max_{\{p:\, p\geq p(r),\, D_C(p,r)\leq d\}} p\, \frac{\log_2 p}{1-r}\, r \\ &\geq \max_{\{p:\, p\geq p(r),\, (\log_2 p+\log_2\log_2 p)/(1-r)+\gamma(r)\leq d\}} p\, \frac{\log_2 p}{1-r}\, r \\ &= \max_{\{p:\, p\geq p(r),\, p\log_2 p\leq 2^{(d-\gamma(r))(1-r)}\}} p\, \frac{\log_2 p}{1-r}\, r \end{aligned}$$

Clearly, there exists an integer $d(r)$ such that for all $d > d(r)$, the constraint set in the last line above is not empty. It follows that for $d > d(r)$, the maximum is attained by the largest integer in the constraint set, denoted p_0, and consequently

$$T_C(r,d) \geq \frac{r}{1-r}\, p_0 \log_2 p_0$$

To simplify this further, let $\hat{p}$ be the solution (possibly non-integer) to the equation $p\log_2 p = 2^{(d-\gamma(r))(1-r)}$. Then, since $h(p) \equiv p\log_2 p$ is a strictly increasing function of p (for $p \geq 1$), it follows that $p_0 = \lfloor \hat{p} \rfloor$, and hence, $h(p_0) \geq h(\hat{p}-1)$. Now, since $h(p)$ is a convex $\cup$ function, $h(\hat{p}) - h(\hat{p}-1) \leq h'(\hat{p}) = \log_2 e + \log_2 \hat{p}$. But, $\log_2 \hat{p} \leq \log_2 h(\hat{p}) = (d-\gamma(r))(1-r)$. Hence, $h(\hat{p}-1) \geq h(\hat{p}) - (d-\gamma(r))(1-r) - \log_2 e$, which shows that

$$T_C(r,d) \geq \frac{r}{1-r} h(p_0) \geq \frac{r}{1-r} h(\hat{p}) - r(d-\gamma(r)) - \frac{r}{1-r}\log_2 e$$

thus proving the theorem. ■

The above lower bound to $T_C(r,d)$ is plotted in Figure 5, without the $o_d(1)$ correction factor. The following corollary is an immediate consequence of the above bound, and the upper bound to $T(r,d)$ derived in Theorem 1.

Corollary 9: For $r \in (0,1)$,

$$\lim_{d\to\infty} \frac{\log_2 T_C(r,d)}{d} = 1-r$$

Also, note that the bounds derived in Theorems 6 and 8 are also bounds to the functions $D(p,r)$ and $T(r,d)$, respectively. Thus, these theorems along with Theorem 1 lead to the following result.

Corollary 10: For $r \in (0,1)$,

$$\lim_{p\to\infty} \frac{D(p,r)}{\log_2 p} = \frac{1}{1-r}$$

$$\lim_{d\to\infty} \frac{\log_2 T(r,d)}{d} = 1-r$$

V. Sync-Timing Codes with Embedded Time Indices

With the cascaded codes described in the previous section, marker indices and data are separately encoded with codes of length m_2 and $L = n - m_1 - m_2 - 3$, respectively. At the expense of an increase in encoding and decoding complexity, somewhat better performance can be obtained by jointly encoding the marker indices with the data. For instance, consider a code that encodes a block of k data bits and an integer from $\{1,2,\ldots,p\}$ into a block of n bits, consisting of the all-ones flag of length m_1 followed by a binary codeword of length $n-m_1$ that begins and ends with a zero and does not contain the flag. Since the maximum number of such codewords is $g(m_1, n-m_1-2)$, we must have $p\,2^k \leq g(m_1, n-m_1-2)$. Therefore, given positive integers p, m_1 and n, to maximize the rate of the resulting code, we choose $k = \lfloor \log_2(g(m_1, n-m_1-2)/p) \rfloor$. As one would expect, when encoding the ith k-block of data, the encoder assigns a codeword of length n to the data block paired with the integer i modulo p. The result is a (p,n) sync-timing code with delay $D = n$, rate

$$R = \frac{k}{n} = \frac{\lfloor \log_2 g(m_1, n - m_1 - 2) - \log_2 p \rfloor}{n}$$

and timing span $T = kp$.

This new class of codes, which we call *prefix-synchronized, embedded-index codes* (PSE codes), can be viewed as a generalization of separable PPS codes, because the latter are the special case where the codebook is the Cartesian product of codebooks for the markers and the data. Non-separable PPS codes are not PSE codes, nor vice versa. It can also be seen that PSE codes are just prefix-synchronized codes in the sense of Gilbert [2], used in a way that yields increased timing span at the expense of reduced rate. The increase in complexity mentioned earlier is due to the fact that encoding and decoding approximately $k + \log_2 p$ bits using one large codebook is more complex than separately encoding k and $\lceil \log_2 p \rceil$ bits using two smaller codebooks.

From values of the function $g(m_1, L)$, it is possible to compute $D_{PSE}(p, r)$, the least delay among all PSE codes with period p and rate at least r. For $r = 0.9$, and various values of p, Figure 6 plots points (d, t), where $d = D_{PSE}(p, r)$ and $t = T_{PSE}(p, r, d)$, which is the maximum timing span of PSE codes with period p, rate at least r, and delay at most d. It can be seen that at this rate, the timing span of PSE codes is larger than that of cascaded codes by a factor of approximately 2–4.

One may improve PSE codes somewhat by considering $1^{m_1}0$ to be the flag, and requiring the codewords that follow it to be sequences of length $n - m_1 - 1$ that do not contain $1^{m_1}0$. The new constraint on the codewords is weaker than the original constraints, as a result of which, as shown in Appendix D, there can be approximately twice as many codewords in the new code as there were in the original PSE code. Thus, at a given rate and delay, the improved PSE codes can have approximately twice the period p, and hence, approximately twice the timing span of the original PSE code.

Finally, one may dispense entirely with markers and flags by deriving (p, n) sync-timing codes directly from comma-free codes [5],[6]. A *comma-free code* is a block code, with a codebook consisting of codewords of length n, having the property that if one extracts any segment of $2n - 1$ successive bits from an arbitrary concatenation of codewords, one can always find the codeword boundaries in this segment. Such codes have long been proposed for conventional synchronization. A comma-free code can be used to obtain a (p, n) sync-timing code by allowing

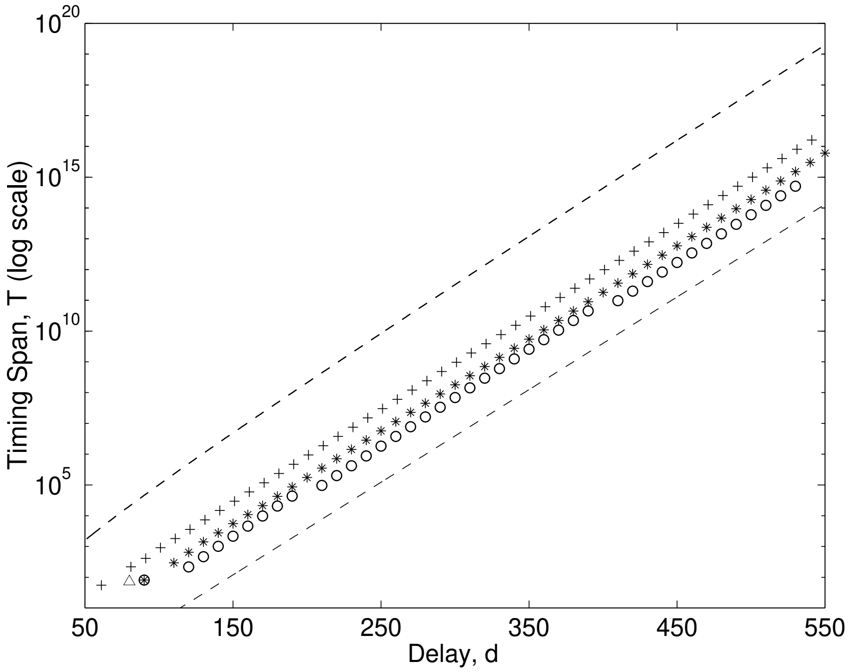

Fig. 6. Timing span vs. delay for cascaded codes, PSE codes and CFST codes with rate at least 0.9. The circles plot $T_C(p,r,d)$ for $d = D_C(p,r)$; the asterisks plot $T_{PSE}(p,r,d)$ for $d = D_{PSE}(p,r)$; the plus signs plot $T_{CFST}(p,r,d)$ for $d = D_{CFST}(p,r)$; the dashed line at the top is the upper bound to $T(r,d)$, and the one at the bottom is the lower bound to $T_C(r,d)$. In each of the above cases, $r = 0.9$.

each codeword to encode a block of k data bits and an integer from $\{1, 2, \ldots, p\}$, as with PSE codes. In this way, one obtains a *comma-free sync-timing code* (CFST code) with delay $D = n$, rate $R = k/n$, and timing span $T = kp$.

This new class of codes is a direct generalization of PSE codes. For a given n, an upper bound for the maximal size of a comma-free codebook can be found in [8]. In [7], it is shown that this upper bound can actually be achieved for odd values of n. From this, it is easy to determine the rate of the best possible (p, n) CFST code, and hence, to compute $D_{CFST}(p,r)$, which is defined as for PSE codes. For various values of p, Figure 6 plots $T_{CFST}(p,r,d)$, which is defined as usual, for $d = D_{CFST}(p,r)$, and $r = 0.9$. It can be seen that, at this rate, CFST codes gain an additional factor of approximately 5 over PSE codes.

The gain in performance in going from PSE codes to CFST codes comes at the expense of an increase in complexity, which is two-fold. Firstly, in order to acquire synchronization, one has to slide a window of width n across the received sequence of encoded bits so as to be able to match the windowed bits with some CFST codeword, while for PSE codes, it is sufficient to have a window of width $m_1 \ll n$, and match the windowed bits with the all-ones flag. Secondly, for

CFST codes, blocks of n bits carry information about the encoded data, as opposed to blocks of $n - m_1 - 2$ bits for PSE codes, and so the actual process of encoding and decoding the data bits is slightly more complex.

Finally, it should be noted that the bounds derived in Theorem 1 for PPS codes are also valid for PSE and CFST codes. These bounds are derived by noting that the redundancy in a codeword of a (p, n) PPS code with rate R is $n(1 - R)$, and since this redundancy encodes an index from $\{1, 2, \ldots, p\}$, we must have $n(1 - R) \geq \log_2 p$. The same reasoning applies to PSE and CFST codes, as well. Also, since cascaded codes form a subset of PSE codes, which are in turn a subset of CFST codes, it follows that the bounds derived in Theorems 6 and 8 also hold for PSE and CFST codes. The dashed lines in Figure 6 show the upper bound to $T(r, d)$ derived in Theorem 1, and the lower bound to $T_C(r, d)$ derived in Theorem 8.

VI. Concluding Remarks

As an illustration of what is possible, suppose that sync-timing codes are applied to a compressed video sequence, whose frames contain 352×288 pixels. First, suppose that the video sequence is compressed to $R_s = 0.5$ bpp (bits per pixel) and that the delay D and timing T must each correspond to one video line, so that immediately after receiving the bits describing a line, the decoder can display it with the proper horizontal alignment. Since timing span need only be as large as delay, one can use conventional prefix synchronized codes (*i.e.*, $p = 1$), and as shown in the first line of Table I, the largest possible rate is 0.95, which implies that 9 redundant bits are added to each line. Second, suppose that while delay should again correspond to one line, timing span should correspond to one frame, so that after receiving the bits describing a line, the decoder can display it in the proper vertical as well as horizontal position. The second line of Table I gives the maximum possible rate in this case, and the corresponding numbers of redundant bits per line, assuming that cascaded codes are used. This rate is achieved by a (288,7,9,196) cascaded code. Note that the period of the cascaded code is $p = 288$, *i.e.*, the number of lines per frame. We see that the rate decreases from 0.95 (5% redundancy) to 0.90 (10% redundancy). Thirdly, suppose one is willing to wait for an entire frame to arrive (rather than just a line) before decoding and displaying. In this case, we allow the delay to correspond to one frame, and if we require rate 0.99 (1% redundancy), then as indicated by the third line of Table I, a cascaded code can achieve timing span equivalent to more than 10^{138} years, assuming

TABLE I
SYNC-TIMING CODES FOR A VIDEO SEQUENCE.

compression rate R_s (bits/pixel)	delay D	timing span T	sync-timing code rate R	redundant bits per line
.5	1 line	1 line	.95	9
.5	1 line	1 frame	.90	20
.5	1 frame	$> 10^{138}$ yrs	.99	1.8
.25	1 line	1 line	.92	8
.25	1 line	1 frame	.81	20
.25	1 frame	$> 10^{61}$ yrs	.99	.9
.1	1 line	1 line	.85	6
.1	1 line	1 frame	.66	18
.1	1 frame	$> 10^{15}$ yrs	.99	.4

30 frames per second. This decoder can begin decoding anywhere in the middle of a huge video sequence, and after receiving one frame, it will know which frame it is decoding. The remaining lines of the table repeat the first three lines, but for smaller values of the compression rate. These have the effect of reducing the permissible delay, making it more difficult to obtain large rate and timing span.

In summary, in this paper we have proposed and analyzed several kinds of block sync-timing codes on the basis of their rate (or redundancy), their resynchronization delay, and their timing span. For a given target rate r, natural marker, cascaded, prefix-synchronized embedded-index (PSE), and comma-free sync-timing (CFST) codes all have timing span increasing exponentially with delay d. For the simplest kind, natural marker codes, the dominant term in the exponent is $d(1-r)/2$, while for the other codes, it is twice this. Thus, large timing spans are attainable with only moderate values of delay.

Appendix A

In this appendix, we prove the non-existence of sync-timing codes with non-zero rate, finite delay and infinite timing span. We begin by introducing some preliminary notation. Given a set $\mathbf{A}$ and a positive integer n, define $\mathbf{A}^n$ to be the set of all length-n sequences of elements in $\mathbf{A}$, $\mathbf{A}^*$ to be the set of all finite sequences (including the empty sequence) of elements in $\mathbf{A}$, and $\mathbf{A}^\infty$ to be the set of all infinite sequences of elements in $\mathbf{A}$.

Let $\mathbf{A}$ denote a finite alphabet of data symbols, and let $\mathbf{N}$ denote the set of non-negative integers. A (binary) sync-timing code operating on $\mathbf{A}$ consists of an encoder and a decoder. The encoder is a one-to-one map $e : \mathbf{A}^\infty \to \{0,1\}^\infty$. The decoder, $d : \{0,1\}^\infty \to (\mathbf{A} \times \mathbf{N})^\infty$, is defined via a sequence of functions $d_n : \{0,1\}^n \to (\mathbf{A} \times \mathbf{N})^* \cup (\mathbf{A} \times \mathbf{N})^\infty$, $n = 1, 2, \ldots$, as follows: for each $(b_1, b_2, \ldots) \in \{0,1\}^\infty$, $d(b_1, b_2, \ldots)$ is given by the concatenation of the sequences $d_n(b_1, b_2, \ldots, b_n)$, $n = 1, 2, \ldots$. In other words, d_n determines the decoder output after b_n has been received, but before b_{n+1} arrives at the decoder. We let (e, d) denote the sync-timing code with encoder e and decoder d.

To facilitate the discussion that follows, given a binary sequence $(b_1, b_2, \ldots)$ and an integer $j > 0$, we use $d^j(b_1, b_2, \ldots, b_j)$ to denote the concatenation of $d_n(b_1, b_2, \ldots, b_n)$, $n = 1, 2, \ldots, j$, and $d_{j+1}^\infty(b_1, b_2, \ldots)$ to denote the concatenation of $d_n(b_1, b_2, \ldots, b_n)$, $n = j+1, j+2, \ldots$. Also, given $\underline{x} \in \mathbf{A}^\infty$ and positive integers i, j, with $i \leq j$, if $e(\underline{x}) = (b_1, b_2, \ldots)$, then we use $e(\underline{x})_i^j$ to denote the sequence $(b_i, b_{i+1}, \ldots, b_j)$. This notation is extended in the obvious way to include the case $j = \infty$.

We are now in a position to provide a definition of a sync-timing code with perfect timing.

Definition: A sync-timing code (e, d) has *perfect timing* (for the switch channel) if and only if for each $\underline{x} = (x_1, x_2, \ldots) \in \mathbf{A}^\infty$ and $n \in \mathbf{N}$, there exists an integer i such that $d(e(\underline{x})_{n+1}^\infty)$ contains $(x_i, i), (x_{i+1}, i+1), \ldots$ as a suffix.

We next extend this definition to include the concept of finite delay.

Definition: A sync-timing code (e, d) with perfect timing has *finite delay* if and only if there exists an integer $D > 0$ such that for each $\underline{x} = (x_1, x_2, \ldots) \in \mathbf{A}^\infty$ and $n \in \mathbf{N}$, there exists an integer i such that some (non-empty) suffix of $d^D(e(\underline{x})_{n+1}^{n+D})$, when concatenated with $d_{D+1}^\infty(e(\underline{x})_{n+1}^\infty)$, forms the sequence $(x_i, i), (x_{i+1}, i+1), \ldots$.

It must be noted that, in the above definitions, the integer i may depend on $\underline{x}$ and n. Note also that the definition of delay D here is not quite the same as in Section II. However, if a code has finite delay in one sense, then it has finite delay in the other.

The final step in this development is to introduce the idea of rate of a sync-timing code. Instead of providing a definition for rate, we propose the following necessary condition for a sync-timing code (e, d) with perfect timing and finite delay to have rate R: there exists an integer $D > 0$ and an integer $c \geq 0$ such that for each $\underline{x} = (x_1, x_2, \ldots) \in \mathbf{A}^\infty$ and $n \in \mathbf{N}$, there exists an integer $i \in [(n+1)R - c, (n+D)R + c]$ such that some (non-empty) suffix of $d^D(e(\underline{x})_{n+1}^{n+D})$, when concatenated with $d_{D+1}^\infty(e(\underline{x})_{n+1}^\infty)$, forms the sequence $(x_i, i), (x_{i+1}, i+1), \ldots$. Again, i may depend on $\underline{x}$ and n. The imposition of this condition is motivated by the fact that any reasonable sync-timing code with perfect timing and finite delay that is also fixed-rate in the sense of Section I would possess this property.

Theorem A.1: There exists no sync-timing code with perfect timing, finite delay and rate $R > 0$.

Since the above shows there is no suitable code for the switch channel, there can be none for the more general channel that permits arbitrary insertion, deletion and substitution errors.

Proof: Suppose there exists such a code (e, d). Let D and c be as above. We first show that, for each integer $T \geq DR + c$, there exists an $\underline{x} = (x_1, x_2, \ldots) \in \mathbf{A}^\infty$ and an $n \in \mathbf{N}$ such that $d^D(e(\underline{x})_{n+1}^{n+D})$ is a finite sequence ending in $(x_t, t), (x_{t+1}, t+1), \ldots, (x_{t+k}, t+k)$, for some integer $t \in [T - RD - 2c, T]$ and some $k \in \mathbf{N}$.

Fix an integer $T \geq DR + c$, and let $n = \lfloor (T-c)/R - D \rfloor$. Let $\underline{x}^{(l)} = (x_1^{(l)}, x_2^{(l)}, \ldots)$, $l = 1, 2, \ldots, 2^D + 1$, be distinct sequences in $\mathbf{A}^\infty$, all of which have a common prefix of length $T - 1$. Let $(b_1^{(l)}, b_2^{(l)}, \ldots) = e(\underline{x}^{(l)})$. Suppose that $d^D(b_{n+1}^{(l)}, \ldots, b_{n+D}^{(l)})$ is an infinite sequence for each l. The necessary condition for the existence of the code in question then shows that for each l, there exists a $t \in [(n+1)R - c, (n+D)R + c] \subset [T - RD - 2c, T]$ such that $d^D(b_{n+1}^{(l)}, \ldots, b_{n+D}^{(l)})$ has $(x_t^{(l)}, t), (x_{t+1}^{(l)}, t+1), \ldots$ as a suffix. But since $t \leq T$ for each l, $d^D(b_{n+1}^{(l)}, \ldots, b_{n+D}^{(l)})$ in fact has $(x_T^{(l)}, T), (x_{T+1}^{(l)}, T+1), \ldots$ as a suffix. Therefore, by the choice of the $\underline{x}^{(l)}$'s, the sequences $d^D(b_{n+1}^{(l)}, \ldots, b_{n+D}^{(l)})$, $l = 1, \ldots, 2^D + 1$, are all distinct. Hence, the sequences $(b_{n+1}^{(l)}, \ldots, b_{n+D}^{(l)})$, $l = 1, \ldots, 2^D + 1$, are all distinct, which is a contradiction because there can be at most 2^D distinct binary sequences of length D. Thus, there exists at least one l such that $d^D(b_{n+1}^{(l)}, \ldots, b_{n+D}^{(l)})$ is a finite sequence.

We next produce an infinite number of encoded sequences, each with some substring of length

D, such that the decoder response to each of these substrings is distinct. Let $T_0 = \lceil DR + c \rceil$, and let $\underline{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots) \in \mathbf{A}^\infty$ and $n_0 \in \mathbf{N}$ be such that if $e(\underline{x}^{(0)}) = \underline{b}^{(0)} = (b_1^{(0)}, b_2^{(0)}, \ldots)$, then $d^D(b_{n_0+1}^{(0)}, \ldots, b_{n_0+D}^{(0)})$ ends in $(x_{t_0}^{(0)}, t_0), \ldots, (x_{t_0+k_0}^{(0)}, t_0 + k_0)$ for some $t_0 \in [T_0 - RD - 2c, T_0]$ and $k_0 \in \mathbf{N}$. Note that the last time index produced by the decoder in response to the length-D substring of $\underline{b}^{(0)}$ beginning at the $(n_0 + 1)$th bit, does not exceed $T_0 + k_0$.

Next, let $T_1 = T_0 + k_0 + \lceil RD + 2c \rceil + 1$, and let $\underline{x}^{(1)}$, $\underline{b}^{(1)}$, n_1, t_1 and k_1 be the analogues of $\underline{x}^{(0)}$, $\underline{b}^{(0)}$, n_0, t_0 and k_0, respectively. Note that $t_1 \in [T_1 - RD - 2c, T_1] \subset [T_0 + k_0 + 1, T_1]$, and hence, the last time index produced by the decoder in response to the substring $(b_{n_1+1}^{(1)}, \ldots, b_{n_1+D}^{(1)})$ is larger than $T_0 + k_0$, but does not exceed $T_1 + k_1$.

Continuing in this manner, for each $l = 1, 2, \ldots$, we can find a sequence $\underline{b}^{(l)} \in e(\mathbf{A}^\infty)$, and integers $T_l, n_l, k_l \in \mathbf{N}$ such that the last time index produced by the decoder in response to the substring $(b_{n_l+1}^{(l)}, \ldots, b_{n_l+D}^{(l)})$ is larger than $T_{l-1} + k_{l-1}$, but does not exceed $T_l + k_l$.

Thus, the sequences $d^D(b_{n_l+1}^{(l)}, \ldots, b_{n_l+D}^{(l)})$, $l \in \mathbf{N}$, are all distinct. This implies that the sequences $(b_{n_l+1}^{(l)}, \ldots, b_{n_l+D}^{(l)})$, $l \in \mathbf{N}$, are all distinct, *i.e.*, there exist an infinite number of distinct binary sequences of length D, which is a contradiction that proves the theorem. ■

Appendix B

We present here some useful results on $g(m_1, \mathcal{L})$, the number of binary sequences of length $\mathcal{L}$ that do not contain a string of m_1 consecutive ones. The following recursion is well-known and, in particular, can be inferred from the generating functions derived by Gilbert [2], and Guibas and Odlyzko [3]: for $\mathcal{L} > m_1$,

$$g(m_1, \mathcal{L}) = \sum_{l=1}^{m_1} g(m_1, \mathcal{L} - l) \tag{16}$$

From the above recursion, it immediately follows that

$$g(m_1, \mathcal{L} + 1) = 2g(m_1, \mathcal{L}) - g(m_1, \mathcal{L} - m_1) \tag{17}$$

The second recursion is used to derive the following result.

Lemma B.1: For $\mathcal{L} > m_1$, $\quad g(m_1, \mathcal{L}) \geq 2^{\mathcal{L}} - (\mathcal{L} - m_1 + 2)\, 2^{\mathcal{L} - m_1 - 1}$

Proof: We prove this result by induction. It is easy to see that the bound holds for $\mathcal{L} = m_1 + 1$. Let the bound hold up to some $\mathcal{L} \geq m_1 + 1$. We then have

$$
\begin{aligned}
g(m_1, \mathcal{L}+1) &= 2g(m_1,\mathcal{L}) - g(m_1, \mathcal{L}-m_1) \\
&\geq 2\,(2^{\mathcal{L}} - (\mathcal{L}-m_1+2)2^{\mathcal{L}-m_1-1}) - 2^{\mathcal{L}-m_1} \\
&= 2^{\mathcal{L}+1} - (\mathcal{L}-m_1+3)2^{\mathcal{L}-m_1}
\end{aligned}
$$

The inequality in the above chain follows from the induction hypothesis and the fact that $g(m_1,\mathcal{L}) \leq 2^{\mathcal{L}}$. Thus, the bound holds for $g(m_1, \mathcal{L}+1)$ as well, which proves the lemma by the principle of mathematical induction. ■

The above lemma does not provide a good estimate of $g(m_1,\mathcal{L})$ for all values of $\mathcal{L}$, since the lower bound is negative for $\mathcal{L} > 2^{m_1+1} + m_1 - 2$. However, it does provide a reasonably sharp estimate of $\widehat{R}(m_1,\mathcal{L})$ for small values of $\mathcal{L}$, as shown in the next lemma.

Lemma B.2: If $\mathcal{L} \leq 2^{m_1} + m_1 - 2$, then $g(m_1,\mathcal{L}) \geq 2^{\mathcal{L}-1}$, and hence, $\widehat{R}(m_1,\mathcal{L}) \geq \dfrac{\mathcal{L}-1}{\mathcal{L}}$

Proof: If $\mathcal{L} \leq 2^{m_1} + m_1 - 2$, then

$$2^{\mathcal{L}} - (\mathcal{L}-m_1+2)2^{\mathcal{L}-m_1-1} \geq 2^{\mathcal{L}} - 2^{m_1}.\,2^{\mathcal{L}-m_1-1} = 2^{\mathcal{L}-1}$$

Therefore, by the previous lemma, $g(m_1,\mathcal{L}) \geq 2^{\mathcal{L}-1}$, thus proving the result. ■

We gain an understanding of the asymptotic behavior of $\widehat{R}(m_1,\mathcal{L})$ through the observation that $\lim_{\mathcal{L}\to\infty} \widehat{R}(m_1,\mathcal{L})$ is simply the Shannon capacity of a constrained system of sequences that forbids runs of more than m_1-1 ones. It is known (see *e.g.* [13]) that the capacity of such a system is given by $\log_2 \rho_{m_1}$, where ρ_{m_1} is the largest real root of the equation $z^{m_1} - z^{m_1-1} - \ldots - z - 1 = 0$. It is shown in [14] that $2(1-2^{-m_1}) \leq \rho_{m_1} < 2$, and hence, $\lim_{m_1\to\infty} \log_2 \rho_{m_1} = 1$. To summarize,

Lemma B.3: For any integer $m_1 \geq 1$,

$$\lim_{\mathcal{L}\to\infty} \widehat{R}(m_1,\mathcal{L}) = \log_2 \rho_{m_1}$$

where ρ_{m_1} is the largest real zero of the polynomial $z^{m_1} - z^{m_1-1} - \ldots - z - 1$. Moreover, $\lim_{m_1\to\infty} \log_2 \rho_{m_1} = 1$.

For the sake of reference, we list the values of $\log_2 \rho_{m_1}$ (rounded to four decimal places) for $m_1 = 1,2,3,4,5$: $\log_2 \rho_1 = 0$, $\log_2 \rho_2 = 0.6942$, $\log_2 \rho_3 = 0.8791$, $\log_2 \rho_4 = 0.9468$ and $\log_2 \rho_5 = 0.9752$.

Finally, we note that Lemma B.3 also follows immediately from the well-known fact (see *e.g.* [14]) that the solution to the recursion in (16) is of the form

$$g(m_1, \mathcal{L}) = \alpha \left(\rho_{m_1}\right)^{\mathcal{L}} (1 + o(1)) \tag{18}$$

where α is a constant and $o(1)$ is a term that goes to zero as $\mathcal{L}$ goes to infinity. This fact is of independent interest, and is used extensively in Appendix D.

Appendix C

In this appendix, we show that given an $r \in (0,1)$ and an integer $p > 0$, for all sufficiently large integers d, there exists a PPS code with period p, rate $R \geq r$, delay $D = d$ and timing span, $T \geq p\,d\,r$, *i.e.*, $T(p,r,d) \geq p\,d\,r$.

For this purpose, we use the family of separable PPS codes called generalized natural marker codes introduced in Section III. Given p, n, and an integer $m_1 > \lceil \log_2 p \rceil$, we consider the (p, m_1, n) generalized natural marker code whose ith marker, M_i, consists of the flag 1^{m_1}, followed by a zero, followed by a binary representation, in $\lceil \log_2 p \rceil$ bits, of the index i. The base codebook consists of $2^{\lfloor \log_2 g(m_1, L-2) \rfloor}$ sequences of length $L = n - m_1 - \lceil \log_2 p \rceil - 1$ that begin and end with a zero, and do not contain m_1 consecutive ones. The rate of this code is given by

$$R_N(p, m_1, n) = \widehat{R}(m_1, n - m_1 - \lceil \log_2 p \rceil - 3)\, \frac{n - m_1 - \lceil \log_2 p \rceil - 3}{n}$$

where $\widehat{R}$ is as defined in Section III.

Lemma B.3 shows that one may choose m_1 and n so as to make the first term in the above arbitrarily close to one. One may also choose m_1 and n to make the second term arbitrarily close to one. Indeed, one may choose m_1 and $d(r,p)$, so that the product of both terms is at least r for all $n \geq d(r,p)$. Thus for all integers $d \geq d(r,p)$, there is a (p, m_1, d) generalized natural marker code with delay $D = d$, rate $R \geq r$, and timing span

$$T = p\, R_N(p, m_1, d)\, d \geq p\, r\, d$$

We have thus proved:

Proposition C.1: Given an integer $p > 0$ and $r \in (0,1)$, there exists an integer $d(r,p)$ such that for all integers $d > d(r,p)$, there exists a generalized natural marker code with period p, delay d and rate at least r, *i.e.*, $T(p,r,d) \geq p\, r\, d$.

In fact, the above argument can be modified slightly to show that for all sufficiently large p, there exists $d(r,p)$ such that for natural marker codes in the sense of Section III, $T_N(p,r,d) \geq p\,r\,d$ for all integers $d \geq d(r,p)$.

Appendix D

As stated in Section V, PSE codes can be improved by considering $1^{m_1}0$ to be the flag instead of 1^{m_1}, and requiring the codewords that follow the flag to be sequences of length $n-m_1-1$ that do not contain $1^{m_1}0$. To quantify this improvement, define $\widetilde{g}(m_1,\mathcal{L})$ to be the set of binary sequences of length $\mathcal{L}$ that do not contain $1^{m_1}0$. Thus, the number of sequences that satisfy the requirements for being a codeword in the improved version of PSE codes is

$$\widetilde{g}(m_1, n-m_1-1) \geq g(m_1, n-m_1-1) \approx 2\,g(m_1, n-m_1-2) \tag{19}$$

The inequality above follows from the fact that the constraint governing the definition of $\widetilde{g}(m_1,\mathcal{L})$ is weaker than that for $g(m_1,\mathcal{L})$, and the approximate equality derives from (18) in Appendix B using the fact that $\log_2 \rho_{m_1} \approx 1$ for $m_1 \geq 4$. This shows that improved PSE codes can, in effect, have at least twice the period p of the original PSE codes at the same rate and delay, and consequently, at least twice the timing span.

To see that the increase in p cannot be substantially larger than a doubling, we first observe that for $\mathcal{L} > m_1$, the set of sequences counted by $\widetilde{g}(m_1,\mathcal{L})$ consists of the $g(m_1,\mathcal{L})$ sequences of length $\mathcal{L}$ that do not contain 1^{m_1}, along with the $g(m_1,\mathcal{L}-(j+1))$ sequences of length $\mathcal{L}$ that end in 01^j, $j = m_1, m_1+1, \ldots, \mathcal{L}-1$, but do not contain 1^{m_1} elsewhere. Thus, we have

$$\begin{aligned}
\widetilde{g}(m_1,\mathcal{L}) &= g(m_1,\mathcal{L}) + \sum_{j=m_1}^{\mathcal{L}-1} g(m_1, \mathcal{L}-(j+1)) \\
&\approx \alpha(\rho_{m_1})^{\mathcal{L}} + \sum_{j=m_1}^{\mathcal{L}-1} \alpha(\rho_{m_1})^{\mathcal{L}-j-1} \\
&= \alpha(\rho_{m_1})^{\mathcal{L}} + \alpha\frac{(\rho_{m_1})^{\mathcal{L}-m_1}-1}{\rho_{m_1}-1} \\
&= \alpha(\rho_{m_1})^{\mathcal{L}}\left(1 + \frac{(\rho_{m_1})^{-m_1}-(\rho_{m_1})^{-\mathcal{L}}}{\rho_{m_1}-1}\right)
\end{aligned}$$

$$\approx \quad g(m_1, \mathcal{L}) \left(1 + \frac{(\rho_{m_1})^{-m_1} - (\rho_{m_1})^{-\mathcal{L}}}{\rho_{m_1} - 1} \right) \tag{20}$$

with the approximate equalities arising from (18). Thus, for $m_1 \geq 4$, we see that $\tilde{g}(m_1, \mathcal{L})$ is only marginally larger than $g(m_1, \mathcal{L})$, and hence in (19), $\tilde{g}(m_1, n - m_1 - 1)$ can only be marginally larger than $2\,g(m_1, n - m_1 - 2)$. Thus, for a given rate and delay, the improved PSE codes have approximately twice the timing span of the original PSE codes. In a sense, eliminating the need for a zero at the end of each codeword makes room for an additional bit that can go into encoding the time index.

Though not quite as simple, a similar approach can be used to improve natural marker codes. As a starting point, consider a conventional natural marker code with blocklength n, period p, flag length $m_1 = \lceil \log_2 p \rceil + 1$, total marker length $m = 2m_1$, rate at least r, and base codewords of length $L = n - m$ that begin and end with a zero and do not contain the flag 1^{m_1}. As with improved PSE codes, one can instead consider $1^{m_1}0$ to be the flag, and choose a base codebook that consists of codewords of length $n - m$ that begin (but not necessarily end) with a zero and do not contain $1^{m_1}0$. Since there are $\tilde{g}(m_1, n - m - 1)$ sequences that satisfy the conditions for inclusion in this new base codebook, and as explained above $\tilde{g}(m_1, n{-}m{-}1) \approx 2\,g(m_1, n{-}m{-}2)$, the new codebook is approximately twice as large as that of the original natural marker code, resulting in an increase in the rate. However, if our goal is simply to have rate at least r, then we could instead choose the codewords in the new base codebook to have length $n - m - 1$. Now, (20) indicates that this new codebook has approximately the same size as the original natural marker codebook. The one bit saved in the length of the base codewords can be given to the index representation, which permits a doubling of p, which implies a doubling of the timing span. However, adding one bit to the index necessitates adding an extra bit to the flag, so as to prevent the flag from appearing in the index. Thus, the net effect is to increase the blocklength n by 1, which decreases the rate by a factor of n/(n+1). If, as sometimes happens, this causes the rate to drop below r, the length of the base codewords, and hence the overall blocklength, will have to be increased to bring it back up to r. However, the required increase in blocklength (*i.e.*, delay) is small relative to the increase in blocklength that would be required to double the timing span according to the approximation $T \sim 2^{d(1-r)/2}$ suggested by Corollary 5. Thus, generally speaking, this improved version of natural marker codes attains approximately twice the timing span, with little effect on rate or delay.

Acknowledgment

The authors would like to thank an anonymous reviewer for pointing out the fact that the performance of natural marker and PSE codes can be improved by considering $1^{m_1}0$ to be the flag instead of 1^{m_1}, as explained in Appendix D.

References

[1] B.G. Haskell, A.P. Puri, and A.N. Netravali, *Digital Video: An Introduction to MPEG-2*, New York: Chapman and Hall, 1997.

[2] E.N. Gilbert, "Synchronization of binary messages," *IRE Trans. Inform. Theory*, vol. IT-6, pp. 470–477, 1960.

[3] L.J. Guibas and A.M. Odlyzko, "Maximal prefix-synchronized codes," *SIAM J. Appl. Math.*, vol. 35, no. 2, pp. 401–418, Sept. 1978.

[4] H. Morita, A.J. van Wijngaarden, and A.J. Han Vinck, "On the construction of maximal prefix-synchronized codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 2158–2166, Nov. 1996.

[5] S.W. Golomb, L.R. Welch, and M. Delbruck, "Construction and properties of comma-free codes," *Biol. Medd. Dan. Vid. Selsk.*, vol. 23, no. 9, 1958.

[6] S.W. Golomb, B. Gordon, and L.R. Welch, "Comma-free codes," *Can. J. Math.*, vol. 10, pp. 202-209, 1958.

[7] W.L. Eastman, "On the construction of comma-free codes," *IEEE Trans. Inform. Theory*, vol. 11, pp. 263–266, Apr. 1965.

[8] S.W. Golomb and B. Gordon, "Codes with bounded synchronization delay," *Inform. Contr.*, vol. 8, pp. 355–372, Aug. 1965.

[9] M.Y. Cheung and J. Vaisey, "A comparison of scalar quantization strategies for noisy channel data transmission," *IEEE Trans. Commun.*, vol. 43, no. 2–4, pp. 738–742, Feb–Apr. 1995.

[10] R.A. Scholtz, "Frame synchronization techniques," *IEEE Trans. Commun.*, vol. 28, no. 8, pp. 1204–1212, Aug. 1980.

[11] N. Kashyap and D.L. Neuhoff, "Periodic prefix-synchronized codes: a generating function approach," paper in preparation.

[12] W.H. Kautz, "Fibonacci codes for synchronization control," *IEEE Trans. Inform. Theory*, vol. 11, pp. 284–292, Apr. 1965.

[13] J.J. Ashley and P.H. Siegel, "A note on the Shannon capacity of run-length-limited codes," *IEEE Trans. Inform. Theory.* vol. 33, pp. 601–605, July 1987.

[14] D.A. Wolfram, "Solving generalized Fibonacci recurrences," *The Fibonacci Quarterly*, vol. 36.2, pp. 129–145, May 1998.