Interactive Low-complexity Codes for Synchronization from Deletions and Insertions

Ramji Venkataramanan
Dept. of Electrical Engineering
Yale University, USA
Email: ramji.venkataramanan@yale.edu

Hao Zhang, Kannan Ramchandran
Dept. of Electrical Engineering & Computer Sciences
University of California, Berkeley, USA
Email: {zhanghao,kannanr}@eecs.berkeley.edu

***Abstract*—We study the problem of synchronization of two remotely located data sources, which are mis-synchronized due to deletions and insertions. This is an important problem since a small number of synchronization errors can induce a large Hamming distance between the two sources. The goal is to effect synchronization with the rate-efficient use of lossless bidirectional links between the two sources. In this work, we focus on the following model. A binary sequence $\mathbf{X}$ of length n is edited to generate the sequence at the remote end, say $\mathbf{Y}$, where the editing involves random deletions and insertions, possibly in small bursts. The problem is to synchronize $\mathbf{Y}$ with $\mathbf{X}$ with minimal exchange of information (in terms of both the average communication rate and the average number of interactive rounds of communication).**

We focus here on the case where the number of edits is much smaller than n, and propose an interactive algorithm which is computationally simple and has near-optimal communication complexity. Our algorithm works by efficiently splitting the source sequence into pieces containing either just a single deletion/insertion or a single burst deletion/insertion. Each of these pieces is then synchronized using an optimal one-way synchronization code, based on the single-deletion correcting channel codes of Varshamov and Tenengolts (VT codes).

I. Introduction

Consider Alice and Bob observing two distributed sources $\mathbf{X}$ and $\mathbf{Y}$, respectively. $\mathbf{Y}$ is an *edited* version of $\mathbf{X}$ where the edits consist of deletions and insertions. Under communication rate constraints, Bob would like to reconstruct Alice's sequence from $\mathbf{Y}$ using minimal communication between him and Alice. Here is an example:

$$\begin{cases} \mathbf{X} = \ldots abraca\ dabradum\ dumdum\ ababab, \ldots \\ \mathbf{Y} = \ldots abacad\ dabradum\ ababab, \ldots . \end{cases} \quad (1)$$

In this case, $\mathbf{Y}$ is obtained from $\mathbf{X}$ by a single deletion of the third letter 'r', followed by a single insertion of a 'd' following the original sixth letter, and followed by a *burst* deletion of the letters '$dumdum$'. Bob wants to reconstruct $\hat{\mathbf{X}}$ from $\mathbf{Y}$ using minimum communication between him and Alice when neither party knows what has been deleted or inserted or the locations of the edits. We will refer to this problem as *synchronization from deletions and insertions*.

There are many motivating scenarios where such a problem needs to be addressed. For instance, in file backup applications, the remotely located data sources often differ only by a small number of deletions and/or insertions, possibly in bursts[1]. It is desirable to have a synchronization tool that achieves successful backup by transferring minimal information. The problem of synchronization also arises in other applications such as file sharing and online file editing. An interesting and important question to ask is: what is the minimal communication rate needed to achieve synchronization? Further, can we design practical synchronization codes with this rate?

A genie-aided lower bound on the rate required in the above example is obtained by assuming that Alice knows $\mathbf{Y}$ exactly, and hence can send Bob the positions of the deletions and insertions.[2] When there are d deletions and i insertions, this would require a rate of at least $\log_2 \binom{n}{s}/n$, where n is the length of $\mathbf{X}$ and $s = d + i$ is the total number of edits. This lower bound is roughly $\frac{s \log_2 n}{n}$ for $s \ll n$. When $\mathbf{Y}$ is unknown to Alice, the results of Levenshtein [1] and Orlitsky et al. [2] suggest that when s is a constant, there exists a one-way zero-error synchronization code with rate less than $\frac{2s \log_2 n + o(1)}{n}$, which is only twice the lower bound. However, such a code has to be found by exhaustive search and has to be decoded using an maximum-likelihood decoder, which has prohibitive computational complexity.

In this work, we adopt a slightly different philosophy in that we insist on realizable (practical) interactive codes, but relax the requirement of zero-error, and are willing to allow the error to go to zero as the length of the source sequence becomes large. In this paper, we provide error guarantees that decrease polynomially in the problem size. Specifically, we show that when the total number of edits is $o(\frac{n}{\log_2 n})$, [3] one can achieve synchronization at near-optimal communication rates with computationally-efficient codes by allowing a small amount of *interaction* between the encoder and the decoder. Our main contributions are listed below. (To be consistent with the literature, we shall refer to synchronization codes as protocols.)

- We design a bi-directional protocol to synchronize from an arbitrary combination of deletions and insertions. The protocol has the following properties: For s random deletions and insertions and any constant $c \geq 1$, the average communication rate from the encoder (Alice) to

[1] One can loosely construe re-ordering of data chunks as deletions followed by insertions.

[2] We may also need to send the values of the edited symbols, but need not include this to obtain a lower bound.

[3] Recall that a function $f(n)$ is $o(\frac{n}{\log_2 n})$ if $\frac{f(n)}{n/\log_2 n} \to 0$ as $n \to \infty$. Also $o(\frac{n}{\log_2 n})$ is slightly weaker than $o(n)$ and is needed for our proof.

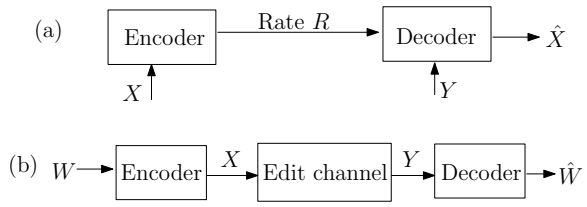

Figure 1. (a) Synchronization: reconstruct X at the decoder using the edited version Y as side-information. (b) Channel coding: send message of rate R through a channel that takes input X and outputs edited version Y.

the decoder (Bob) is $(4c+1)\frac{s\log_2 n}{n}$, the average rate in the reverse direction (from Bob to Alice) is $10\frac{s-1}{n}$, and the probability of error goes to zero as $\frac{d\log_2 n}{n^c}$.

- The average number of rounds of interaction needed by the protocol is approximately $2\log_2 s$. For applications that have a hard bound on the number of rounds of interaction, we show how the protocol can be adapted to trade-off higher communication rate for fewer rounds of interaction.
- The protocol has $O(n)$, i.e. linear, computational complexity and is hence practically feasible.
- The protocol can also efficiently handle deletions and insertions that occur in bursts without having prior knowledge of the burst lengths. In this paper, we consider short bursts ($o(1)$ length) since we are in the regime of $o(n)$ edits. However, our protocol can be extended to efficiently handle larger bursts of length $\Theta(n)$ [4]. This is ongoing work and will be part of a future publication.

The rest of the paper is organized as follows. Section II describes related work. In Section III we review Varshamov Tenengolts (VT) codes [3] (which form a key ingredient of our algorithm), and show how they can be used to synchronize from a single deletion or insertion. In Section IV, we present an interactive protocol to synchronize from multiple deletions and insertions, and give theoretical bounds on its expected communication rate, the number of rounds of interactions and probability of error. In Section V, we describe how to efficiently deal with deletions and insertions that occur in bursts. Section VI presents experimental results for various test cases, and Section VII concludes the paper.

In the sequel, $\mathbf{X}$ always denotes the sequence available at the encoder (Alice) and has length n. $\mathbf{Y}$ is the sequence at the decoder (Bob) which needs to be synchronized to $\mathbf{X}$. To keep the exposition simple, we assume both sequences are binary valued. However, all the proposed protocols can be easily extended to any finite alphabet. We also assume that the lengths of $\mathbf{X}$ and $\mathbf{Y}$ are known to both the encoder and the decoder at the outset. If not, this can be achieved by a one-way transmission of a vanishingly small number of bits: if it is known that the maximum possible number of edits is s_{max}, the decoder can convey its length by sending $\log_2 2s_{max}$ bits.

II. Related Work

As shown in Figure 1, the one-way synchronization problem can be cast as a problem of source coding with side-information at the decoder [4], [5]. It is well-known that source coding with decoder side-information is closely related to channel coding [6], [7]. The channel coding problem corresponding to one-way synchronization is shown in Figure 1(b) - the goal is to reliably communicate over a channel that takes input X and produces an edited version Y as output.

Levenshtein [1] showed that if edit channel in Figure 1(b) is one that introduces at most s edits (insertions + deletions) in an input block of length n, the maximal zero-error rate is bounded in between $[1-\frac{2s\log_2 n+o(1)}{n}, 1-\frac{s\log_2 n+o(1)}{n}]$. (In this result, s is fixed and n grows to ∞.) In [2], an upper bound on the minimum rate required for zero-error source coding with decoder side-information was established in terms of the maximal zero-error rate for the corresponding channel coding problem. Combining these results, one can obtain an upper bound on the minimum rate for the synchronization problem to be $\frac{2s\log_2 n+o(1)}{n}$. This rate is within a factor of two of the fundamental limit: even if the encoder *knew* the positions of the edits a priori, it would need to send $s\left(\log_2 n+o(1)\right)$ bits to indicate these positions. This guarantees the existence of zero-error one-way synchronization codes with near-optimal rates for any fixed s as $n\to\infty$. However, we still do not know of the existence of codes having tractable encoding and decoding complexity.

There is a large body of work dealing with capacity and coding for deletion and insertion channels (see [8] and references therein). In particular, concatenated codes for channels with deletions and insertions are constructed in [9], where a 'watermark' inner code detects the positions of insertions and deletions and provides soft outputs to an outer Low-Density-Parity-Check (LDPC) code. One idea for designing one-way synchronization codes is to mimic the construction in [9] by letting the encoder send an inner watermark along with syndromes of an outer LDPC code. In this approach, one needs to design a separate synchronization code for each block length n. This is not desirable since we would like a scalable synchronization protocol that works for any n. Further, the channel codes in [9] are designed to correct a large number of edits in short blocks, so they have low rates and relatively high probability of error. In contrast, we consider synchronization from $o(n)$ edits in this work.

In [10], protocols for synchronization under more general edit models were designed with the aim of minimizing communication complexity; computational efficiency was not the main objective. There also exist many practical synchronization tools. For example, rsync [11] is a UNIX utility that does bit-exact synchronization between files. Recently, Zhang, Yeo and Ramchandran introduced VSYNC [12] that targets video applications where synchronization needs to be performed to within some user-defined distortion.

The main idea behind our approach is to use interaction to

[4] In other words, the burst lengths lie between $\alpha_1 n$ and $\alpha_2 n$ for some $0<\alpha_1<\alpha_2<1$

efficiently split the source sequence into pieces containing either: a) only a single deletion/insertion or b) only a single burst deletion/insertion. Each of these pieces is then synchronized using a near-optimal one-way synchronization code.

III. Synchronizing from One Deletion/Insertion

In this section, we describe how to optimally synchronize from a single deletion or insertion. The one-way synchronization protocol for a single deletion is based on the single-deletion correcting channel codes introduced by Varshamov and Tenengolts [3] (henceforth VT codes).

Definition 1. *For $0 \leq a \leq n$, the block length n VT code $VT_a(n)$ consists of all binary vectors $\mathbf{X} = (x_1, \ldots, x_n)$ satisfying*

$$\sum_{i=1}^{n} i x_i \equiv a \ mod\ (n+1). \tag{2}$$

For example, the code $VT_0(4)$ with block length $n = 4$ is

$$\begin{aligned} VT_0(4) &= \{(x_1, x_2, x_3, x_4) : \sum_{i=1}^{4} i x_i \bmod 5 = 0\} \\ &= \{0000, 1001, 0110, 1111\}. \end{aligned} \tag{3}$$

For any $a \in \{0, \ldots, n\}$, the code $VT_a(n)$ can be used to communicate reliably over a channel that introduces at most one deletion in a block of length n. Levenshtein proposed a simple decoding algorithm [1], [13] for a VT code, which we reproduce below. Assume the channel code $VT_a(n)$ is used.

- Suppose a codeword $\mathbf{X} \in VT_a(n)$ is transmitted over the channel, the bit in position p is deleted and $\mathbf{Y}$ is received. Let there be L_0 0's and L_1 1's to the left of the deleted bit, and R_0 0's and R_1 1's to the right of the deleted bit (with $p = 1 + L_0 + L_1$).
- Compute the weight $wt(\mathbf{Y}) = L_1 + R_1$ of $\mathbf{Y}$ and the checksum $\sum_i i y_i$. If the deleted bit is 0, the new checksum is $R_1 (\leq wt(\mathbf{Y}))$ less than it was before. If the deleted bit is 1, the new checksum is $p + R_1 = 1 + L_0 + L_1 + R_1 = 1 + wt(\mathbf{Y}) + L_0 (> wt(\mathbf{Y}))$ less than it was before.
- Hence, if the deficiency in the checksum, say D, is less than or equal to $wt(\mathbf{Y})$ we know that a 0 was deleted, and we restore it just to the left of the rightmost R_1 1's. Otherwise a 1 was deleted and we restore it just to the right of the leftmost L_0 0's.

As an example, assume the code $VT_0(4)$ is used and $\mathbf{X} = (1, 0, 0, 1) \in VT_0(4)$ is transmitted over the channel.

1) If the second bit in $\mathbf{X}$ is deleted and $\mathbf{Y} = (1, 0, 1)$, then the new checksum is 4, and the deficiency $D = 5 - 4 = 1 < wt(\mathbf{Y}) = 2$. The decoder inserts a 0 after $D = 1$ 1's from the right to get $(1, 0, 0, 1)$.
2) If the fourth bit in $\mathbf{X}$ is deleted and $\mathbf{Y} = (1, 0, 0)$, then the new checksum is 1, and the deficiency $D = 5 - 1 = 4 < wt(\mathbf{Y}) = 2$. The decoder inserts an 1 after $D - wt - 1 = 2$ 0's from the left to get $(1, 0, 0, 1)$.

The codes $VT_0(n)$ are optimal single-deletion correcting channel codes for $n \leq 9$. For each n, $VT_0(n)$ has size $\geq \frac{2^n}{n+1}$, and are thus near-optimal for large n [1].

A. One-way Synchronization using VT Syndromes

As also observed in [14], VT codes can be used to synchronize from a single deletion. Length-n sequence $\mathbf{X}$ is available at the encoder; $\mathbf{Y}$, obtained from $\mathbf{X}$ by deleting one bit is available at the decoder. To synchronize a 1-bit deletion, the encoder sends the checksum of its sequence $\mathbf{X}$ modulo $(n+1)$. The decoder receives this value, say a, and decodes its sequence $\mathbf{Y}$ to a codeword in $VT_a(n)$. This codeword is equal to $\mathbf{X}$ since $VT_a(n)$ is a single-deletion correcting channel code.

Since $a \in \{0, \ldots, n\}$, $\log_2(n+1)$ bits of transmission are needed. This is asymptotically optimal - even if the encoder knew which bit was deleted, it has to send at least the position of the deletion using $\log_2 n$ bits. We essentially use the partitioning of the $\{0,1\}^n$ space by the (non-linear) codes $VT_a(n), 1 \leq a \leq n$ to synchronize. This is similar to using cosets (partitions) of a linear code to perform Slepian-Wolf coding. Hence we shall refer to $\sum_i i x_i$ mod $(n+1)$ as the VT-syndrome of $\mathbf{X}$.

If $\mathbf{Y}$ was obtained from $\mathbf{X}$ by a single *insertion*, one can also use a similar algorithm to delete the inserted bit. The only difference is that the decoder now has to use the *excess* in the checksum of $\mathbf{Y}$ and compare it to its weight. In summary, when the edit is either a single deletion or insertion, one can optimally synchronize $\mathbf{Y}$ to $\mathbf{X}$ with a simple one-way algorithm with zero error and no interactions.

IV. Synchronizing from Multiple Deletions and Insertions

In this section, we design an interactive protocol to correct multiple insertions and deletions. We first consider the special case of synchronizing from multiple deletions since it contains most of the key ideas.

A. Deletions Only

The sequence $\mathbf{Y}$ is obtained by deleting $d > 1$ bits from $\mathbf{X}$ where d is much smaller than n. From Section III, we know that if the number of deletions is one, using VT codes is sufficient to achieve synchronization with zero error. Our protocol aims to break down the synchronization problem into sub-problems, each containing only a single deletion. This is achieved efficiently through a 'divide-and-conquer' strategy which uses interactive communication. We now explain the protocol using a simple example. Consider:

$$\begin{aligned} \mathbf{X} &= 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1 \\ \mathbf{Y} &= 1\ 1\ 0\ 0\ \textvisiblespace\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ \textvisiblespace\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ \textvisiblespace \end{aligned}$$

where $\mathbf{Y}$ is obtained from $\mathbf{X}$ from three deletions indicated by "␣" above. As explained before, it is assumed that the number of deletions $d = 3$ is known to both the encoder and the decoder. We propose the following protocol:

- The encoder (Alice) maintains an unresolved list L_X, whose entries consist of the yet-to-be-synchronized substrings of $\mathbf{X}$, and is initialized to be $L_X = \{\mathbf{X}\}$. The decoder (Bob) maintains a corresponding list L_Y.
- In each round, the encoder sends $l = c \log_2 n$ bits around the *center* of each substring in L_X to the decoder,

who tries to align these bits near the center of the corresponding substring in L_Y (c is a constant greater than 1). In the above example, the encoder uses $l = 6$ center bits and sends bits 9 to 14: "1 0 1 0 1 1".

If the decoder successfully matches the center bits, from the position at which alignment happens, it knows the exact number of deletions that happened in the left and right halves of the substring. In the example case, the decoder knows that there is a single deletion on the left and two deletions on the right. For each of these halves:

- If the number of deletions is *zero*, this half has been synchronized. (None of the substrings in the example fall into this case.)
- If the number of deletions is *one*, the decoder requests the VT syndrome of this half for synchronization. (In the example, the decoder requests the VT syndrome of the left half of $\mathbf{X}$ at the end of the first round.)
- If the number of deletions is *greater than one*, the decoder puts this half in L_Y. The encoder puts the corresponding half in L_X.

If one or more of the center-bits are among the deletions, the decoder may not be able to align the received bits close to the center of its substring. In this case, the decoder requests an adjacent set of center bits for this substring in the next round.

- The process continues until L_Y (and L_X) is empty.

We show the following theorem.

Theorem 1. *(Deletions Only) Suppose there are d deletions, where $d \sim o(\frac{n}{\log_2 n})$.[5] The positions of the deletions are random and $l = c\log_2 n$ center-bits are used for alignment each time they are requested. ($c \geq 1$.)*

***(a)** The probability of error , i.e., the probability that the protocol synchronizes incorrectly is at most $\frac{d\log_2 n}{2n^c}$.*

***(b)** If $N_{1\to 2}(d)$ ($N_{2\to 1}(d)$) denote the number of bits transmitted from the encoder (decoder) to the decoder (encoder),*

$$EN_{1\to 2}(d) < (2c+1)\, d\log_2 n, \quad EN_{2\to 1}(d) < 8(d-1).$$

***(c)** The probability that the algorithm terminates after r rounds is at least $(1-(d+1)2^{-r})^d$. In particular, the probability that the protocol has not terminated after $k + 2\log_2 d$ rounds is $2^{-k} + o(2^{-k})$. Consequently, the expected number of rounds taken by the protocol to terminate is approximately $4 + 2\log_2 d$.*

Proof: See Appendix.

Remark: The average communication rate is $(2c+1)\frac{d\log_2 n}{n}$, which is larger than the genie bound of $\frac{d\log_2 n}{n}$ by a constant factor $(2c+1)$. The average feedback rate is proportional to the number of deletions, and is small since d is much smaller than n. This is useful in applications where feedback is expensive.

Experimental results described in Section VI validate that the actual performance is close to the bounds in Theorem 1.

[5]Since the center-bits are used for alignment, we need the probability of a center-bit being deleted to be negligibly small. This is achieved when the number of deletions is $o(\frac{n}{\log_2 n})$, which is slightly smaller than $o(n)$.

B. Deletions + Insertions

Suppose now that $\mathbf{Y}$ is obtained from $\mathbf{X}$ by a combination of d random deletions and i random insertions.

Recall that in the deletions-only case, the decoder uses the offset of the center-bits to determine the number of deletions in the left and right halves of each substring under consideration, and this continues until there exists no more than a single deletion. When there are insertions present, the offset indicates the number of *net* deletions. For example, an offset of zero can be obtained when there is an equal number of deletions and insertions. To distinguish between these cases, we invoke a 'guess-and-check' mechanism by applying a hashing technique whenever the number of net deletions is zero, in order to check whether the two substrings being considered are indeed synchronized (consistent with zero deletions and zero insertions). The overall protocol works in a divide-and-conquer fashion as before, and is described below:

- The encoder maintains an unresolved list L_X, whose entries consist of the yet-to-be-synchronized substrings of $\mathbf{X}$, and is initialized to be $L_X = \{\mathbf{X}\}$. The decoder maintains a corresponding list L_Y.
- In each round, the encoder sends $l = c\log_2 n$ bits around the *center* of each substring in L_X to the decoder, who tries to align these bits near the center of the corresponding substring in L_Y. ($c \geq 1$) For each of these halves:
  - If the net number of deletions is *zero*, the decoder checks if this half has been synchronized by requesting hashes from the encoder. If the hashes agree, declare synchronization; else put this half in L_Y (and correspondingly in L_X).
  - If the net number of deletions or insertions is *one*, the decoder requests the VT syndrome of this half and performs VT decoding. It also requests hashes to verify synchronization. If the hashes agree, declare synchronization; else the decoder puts this half in L_Y (and correspondingly in L_X).
  - If the number of deletions or insertions is *greater than one*, put this half in L_Y (and correspondingly in L_X).
- The process continues until L_Y (or L_X) is empty.

The only difference between this protocol and the one for the deletion-only case is the hash verification process. When there is a net deletion (or insertion) of one, the protocol assumes a single deletion (or insertion), decodes it using the VT syndrome, and uses the hash to verify synchronization. Similarly, when the lengths of two substrings being compared are the same, the hash is used to check if they are identical. In other words, the protocol works in a 'guess-and-check' fashion. The following theorem describes the performance of the protocol when the positions of insertions and deletions are random. Due to space constraints, the proof is omitted and will be presented in an extended version of the paper.

Theorem 2. *(Insertions + Deletions) Suppose there are d deletions and i insertions, and let $s = (d+i) \sim o(\frac{n}{log_2 n})$. The positions of the edits are random, and l center-bits and*

L hash bits are used each time they are requested. For $l = L = c\log_2 n$:

***(a)** The probability of error , i.e., the probability that the protocol synchronizes incorrectly is at most $\frac{s\log_2 n}{n^c}$.*

***(b)** If $N_{1\to2}(s)$ ($N_{2\to1}(s)$) denote the number of bits transmitted from the encoder (decoder) to the decoder (encoder),*

$$EN_{1\to2}(s) < (4c+1)\, s\log_2 n, \quad EN_{2\to1}(s) < 10(s-1)$$

***(c)** The probability that the algorithm terminates after r rounds is at least $(1-(s+1)2^{-r})^s$. In particular, the probability that the algorithm has not terminated after $k + 2\log_2 s$ rounds is $2^{-k} + o(2^{-k})$. Consequently, the expected number of rounds taken by the protocol to terminate is approximately $4 + 2\log_2 s$.*

Remark: To obtain the results of the theorem, we assume a $c\log_2 n$-bit hash with collision probability $\frac{1}{n^c}$. Universal hashing [15], for instance, has this property.

C. Synchronization Protocol

We summarize our proposed synchronization protocol by describing the algorithms at the encoder and the decoder respectively in Algorithm 1 and Algorithm 2.

Algorithm 1 Synchronization Protocol at the Encoder

The encoder keeps an unresolved list L_X which it initializes by setting $L_X = \{\mathbf{X}\}$.
while L_X is non-empty **do**
  Receive from the decoder the instructions I_s for all substrings $s = 1, 2, \ldots, |L_X|$ in L_X, and do the following for all $s \in L_X$ in a single transmission:
  for all substring $s \in L_X$ **do**
    if $I_s =$ "Verify" **then**
      Apply and send the hash of string s.
    else if $I_s =$ "VT mode" **then**
      Send both the VT syndrome and the hash of string s.
    else if $I_s =$ "Centering" **then**
      Send bits around the center of string s.
    else if $I_s =$ "Split" **then**
      Split the original substring into two halves and put both of them into L_X; remove the original string.
    else if $I_s =$ "Matched" **then**
      Remove string s from L_X.
    end if
  end for
end while

Algorithm 2 Synchronization Protocol at the Decoder

The decoder keeps an unresolved list L_Y which it initializes by setting $L_Y = \{\mathbf{Y}\}$.
while L_Y is non-empty **do**
  Read the instructions $I_s, s = 1, 2, \ldots, |L_Y|$ sent to $\mathbf{X}$ in the previous round, and use them with the responses from $\mathbf{X}$ to decide the new set of instructions for each substring $s \in L_Y$ as follows.
  for all strings in L_Y **do**
    if I_s was "Verify" **then**
      Compare the hash of string s with that sent by $\mathbf{X}$. If the hashes match, add instruction "Match"; else add "Centering".
    else if I_s was "VT mode" **then**
      Use the VT syndrome sent by $\mathbf{X}$ to correct the substring by deleting or inserting a single bit from string s and compare the hashes. If the hashes match, add "Match"; else add "Centering".
    else if I_s was "Centering" **then**
      Try to find a substring in the center of string s that matches with that sent by $\mathbf{X}$. If successful, split s into two halves by the center, remove s from L_Y and put the new two substrings into L_Y, and add "Split"; else, request more centering bits by adding the instruction "Centering".
    end if
  end for
  Send the new set of instructions to $\mathbf{X}$.
end while

V. Efficient Synchronization from Bursts

Burst deletions and insertions can be a major source of mis-synchronization in practical applications (particularly involving audio and video edits) and are therefore important to address. The protocol described in the previous section seeks to divide the original string into pieces with one insertion/deletion each. This is not an efficient way of dealing with bursts since a burst consists of multiple edits adjacent to each other. We start with the simple case of a single burst of deletions and present a one-way protocol for synchronization. We then address multiple heterogenous bursts of both deletions and insertions (i.e., the bursts may have different lengths).

A. A Single Burst of Deletions

The encoder has binary string $\mathbf{X}$ of length n from which a single burst of B bits is deleted. The resulting string $\mathbf{Y}$ of length $n - B$ is available at the decoder. The following one-way protocol synchronizes $\mathbf{Y}$ to $\mathbf{X}$.

Encoding: The encoder divides $\mathbf{X}$ into B substrings, where the k^{th} substring $\mathbf{X}^k$ is:

$$\mathbf{X}^k = (x_k, x_{B+k}, x_{2B+k}, \ldots),\ 1 \le k \le B. \tag{4}$$

The length of each of substring is either $\lfloor \frac{n}{B} \rfloor$ or $\lfloor \frac{n}{B} \rfloor + 1$. For simplicity, we assume in the sequel that the length of every substring is $\frac{n}{B}$. The encoder computes the VT syndromes a_k , $1 \le k \le B$ of each substring and sends it to the decoder. This will cost a total of $N_{burst} = B\log_2 \frac{n}{B} + 1$ bits of transmission.

Decoding: The decoder splits $\mathbf{Y}$ it into B substrings $\mathbf{Y}^k$:

$$\mathbf{Y}^k = (y_k, y_{B+k}, y_{2B+k}, \ldots),\ 1 \le k \le B. \tag{5}$$

Observe that each substring $\mathbf{Y}^k$ is obtained by deleting exactly one bit from $\mathbf{X}^k$. Using the syndrome a_k, $\mathbf{X}^k$ can be recovered by decoding $\mathbf{Y}^k$ to the unique codeword in $VT_{a_k}(\frac{n}{B})$. The reconstructed substrings $\mathbf{X}^k$ are then combined appropriately to recover $\mathbf{X}$.

A lower bound on the number of bits needed is obtained by assuming the encoder knows the exact location of the burst deletion. Then it has to send the decoder two pieces of information: a) the location of the starting position of the burst and b) the actual bits that were deleted. This will cost $\log_2(n - B + 1) + B$ bits. Hence the number of bits transmitted by the proposed protocol is at most B times this lower bound.

A similar one-way protocol can be designed to synchronize from a single burst *insertion* (see discussion in Section III).

B. Heterogenous Bursts of Deletions and Insertions

When there are combinations of edits (deletions and insertions), where some edits occur in isolation and others in bursts of varying lengths, we need to efficiently detect the bursts and their lengths. We use a 'guess-and-check' algorithm to achieve this. In particular, when the number of net deletions in a substring does not change after a certain number of rounds (say T_{burst}), we will hypothesize that a burst deletion (or insertion) has occurred. We request VT syndromes and correct the substring assuming a burst and then use the hash to verify the results of correction. If the hash succeeds, we declare that the substring has been synchronized correctly, else we infer that the deletions (or insertion) did not occur in a burst, and continue to split the substring. The value of T_{burst} can be adjusted to trade-off between the number of interactions and communication rate. We omit discussions of the protocol performance here due to space constraints, but provide simulation results in the next section.

VI. Experimental Results

In this section, we validate through simulations that the bounds predicted by theory are tight in practice as well. In all the experiments, $\mathbf{X}$ is an i.i.d Bernoulli(0.5) sequence.

A. Case 1: Random Deletions (No bursts)

$\mathbf{Y}$ is generated by deleting bits randomly from $\mathbf{X}$. No insertions are applied. We test four subcases by setting $(n = 10^6, d = 10)$, $(n = 10^6, d = 100)$, $(n = 10^7, d = 100)$ and $(n = 10^7, d = 1000)$. The number of center-bits used is $l = 20$, which is in between $[\log_2 10^6, \log_2 10^7]$. No hashes are used. Table I(a) shows for each case the number of round-trips, the number of communication bits each way, the total number bits and the probability of error. For each case, the results are averaged over 1000 runs of simulations, and the number of bits are shown as a fraction of n. Table I(b) shows the corresponding theoretical values.

As shown in the table, the simulation results match well with theoretical analysis. Also, the total communication rate is very close to the genie bound. Since the theoretical probability of error is too small, we were not able to detect any errors in the 1000 runs that were tested.

B. Case 2: Random Insertions + Deletions (No bursts)

$\mathbf{Y}$ is generated by randomly deleting and inserting (0's and 1's with equal probability) the same number of bits from $\mathbf{X}$ in a non-burst fashion. We apply a universal hash [15] to check every pair of substrings that has potential matches. The number of center-bits used is $l = 20$, and the number of bits for the universal hash is also $L = 20$. We test four subcases similar to those used in Case 1. Table II lists the results.

As shown in the table, both the number of interactions and the total communication rate increased when there are additional insertions introduced. It is worth noting that the number of interactions increased by only a small amount compared to the deletion only case, since it is logarithmic in the number of edits. The protocol successfully corrected all the deletions and insertions, and the total communication rate is approximately five times the genie bound.

C. Case 3: Burst Deletions

In this case, $\mathbf{X}$ is an i.i.d Bernoulli(0.5) sequence of length $\mathbf{X}$ $n = 10^6$. To generate $\mathbf{Y}$, a number of burst deletions each starting from randomly selected positions are applied to $\mathbf{X}$. The number of bursts is 10 and each burst has length of either $B = 20$ or $B = 100$ with equal probability. We test two subcases by setting $T_{burst} = 2$ and $T_{burst} = \infty$, where infinity means no burst mode will be used. (Recall that if the number of net deletions in a substring does not change after T_{burst} rounds, we hypothesize a burst.) In each case, the results are averaged over 1000 runs of simulations. Table III shows the results.

When $T_{burst} = 2$, we are more aggressive in guessing that the deletions occur in bursts. In this case, we end up spending a few more bits, i.e., a total of $7.9e^{-3}$ compared to $5.8e^{-3}$ in terms of total communication rate. This is because when false positives happen, (i.e., when we assume bursts while they are not), we waste both VT syndrome bits and hash bits in the verification process. However, the number of interactions needed reduces from 16.5 to 9 on average. We will study the tradeoffs between such aggression in hypothesizing a burst and the communication rate in an extended version of the paper.

VII. Discussion

We considered the problem of synchronization from insertions and deletion edits, where the edits may occur in isolation or in bursts. Designing tractable one-way codes for synchronization is very difficult even when the number of edits is small. In this work, we showed that a small amount of interaction can help design low-complexity codes with close to optimal communication rate.

The guiding principle was to use interaction to efficiently isolate segments of the source containing either a single deletion/insertion or a single burst deletion/insertion. The segments with single deletion are then synchronized using VT syndromes without any interaction. We can easily generalize our protocol to work with general alphabets using non-binary VT codes [16].

The protocol can be modified to satisfy various restrictions on the amount of interaction that may be imposed in a practical

Table I

$\mathbf{Y}$ IS GENERATED BY DELETING BITS RANDOMLY FROM $\mathbf{X}$ (NO BURSTS). THE NUMBER OF CENTER-BITS $l = 20$. THE TABLE SHOWS THE NUMBER OF ROUND-TRIPS, THE COMMUNICATION RATE IN EACH DIRECTION, THE TOTAL RATE AND THE PROBABILITY OF ERROR FROM THE SIMULATIONS AND THE BOUNDS OF THEOREM 1. FOR EACH CASE, THE RESULTS ARE AVERAGED OVER 1000 RUNS OF SIMULATIONS.

Config.		No. of rounds		$\mathbf{X} \to \mathbf{Y}$		$\mathbf{Y} \to \mathbf{X}$		Total Comm. rate		Prob. of error	
n	d	Sim.	Theo.	Sim.	Theo.	Sim.	Theo.	Sim.	Genie	Sim.	Theo.
10^6	10	7.9	10.6	$4.3e^{-4}$	$6.0e^{-4}$	$5.6e^{-5}$	$5.7e^{-5}$	$4.9e^{-4}$	$2.0e^{-4}$	0	$9.5e^{-7}$
10^7	100	14.5	17.3	$4.5e^{-4}$	$6.3e^{-4}$	$6.2e^{-5}$	$6.3e^{-5}$	$5.1e^{-4}$	$2.3e^{-4}$	0	$9.5e^{-7}$
10^6	100	14.4	17.3	$4.2e^{-3}$	$6.0e^{-3}$	$6.2e^{-4}$	$6.3e^{-4}$	$4.8e^{-3}$	$2.0e^{-3}$	0	$9.6e^{-7}$
10^7	1000	19.3	23.9	$4.3e^{-3}$	$6.3e^{-3}$	$6.3e^{-4}$	$6.3e^{-4}$	$4.9e^{-3}$	$2.3e^{-3}$	0	$9.6e^{-7}$

Table II

$\mathbf{Y}$ IS GENERATED BY FIRST DELETING BITS RANDOMLY FROM $\mathbf{X}$ FOLLOWED BY RANDOM INSERTIONS OF 0'S AND 1'S WITH EQUAL PROBABILITY (NO BURSTS). THE NUMBER OF INSERTIONS i IS EQUAL TO THE NUMBER OF DELETIONS d. THE NUMBER OF CENTER-BITS $l = 20$, AND THE NUMBER OF BITS FOR THE UNIVERSAL HASH $L = 20$.

Config.		No. of rounds		$\mathbf{X} \to \mathbf{Y}$		$\mathbf{Y} \to \mathbf{X}$		Total Comm. rate		Prob. of error	
n	$d = i$	Sim.	Theo.	Sim.	Theo.	Sim.	Theo.	Sim.	Genie	Sim.	Theo.
10^6	10	10.1	12.6	$1.9e^{-3}$	$2.0e^{-3}$	$1.9e^{-4}$	$1.9e^{-4}$	$2.1e^{-3}$	$4.1e^{-4}$	0	$9.5e^{-7}$
10^7	100	14.9	19.3	$2.1e^{-3}$	$2.1e^{-3}$	$1.6e^{-4}$	$2.0e^{-4}$	$2.3e^{-3}$	$4.8e^{-4}$	0	$9.5e^{-7}$
10^6	100	16.4	19.3	$1.7e^{-2}$	$2.0e^{-2}$	$1.6e^{-3}$	$2.0e^{-3}$	$1.9e^{-2}$	$4.1e^{-3}$	0	$9.6e^{-7}$
10^7	1000	21.3	25.9	$2.0e^{-2}$	$2.1e^{-2}$	$1.6e^{-3}$	$2.0e^{-3}$	$2.2e^{-2}$	$4.8e^{-3}$	0	$9.6e^{-7}$

Table III

$\mathbf{X}$ IS AN IID BERNOULLI(0.5) SEQUENCE OF LENGTH $n = 10^6$. 10 BURST DELETIONS ARE APPLIED TO GET $\mathbf{Y}$, WHERE THE BURST LENGTHS ARE EITHER $B = 20$ OR $B = 100$ WITH EQUAL PROBABILITY. IF THE NUMBER OF NET DELETIONS IN A SUBSTRING DOES NOT CHANGE AFTER T_{burst} ROUNDS, WE HYPOTHESIZE A BURST. WE TEST TWO SUBCASES: $T_{burst} = 2$ AND $T_{burst} = \infty$, WHERE INFINITY MEANS NO BURST MODE WILL BE USED.

T_{burst}	No. of rounds	$\mathbf{X} \to \mathbf{Y}$	$\mathbf{Y} \to \mathbf{X}$	Comm. rate		Prob. of error
	Sim.	Sim.	Sim.	Sim.	Genie	Sim.
∞	16.5	$5.0e^{-3}$	$8.1e^{-4}$	$5.8e^{-3}$	$9.0e^{-4}$	0
2	9.0	$7.7e^{-3}$	$2.4e^{-4}$	$7.9e^{-3}$	$9.0e^{-4}$	0

setting. For instance, there may be a hard bound on the number of rounds of interaction. Recall from Theorem 2 that the average number of rounds for our protocol is roughly $2\log_2 s$, where s is the number of edits. We can modify the protocol as follows to terminate in fewer rounds at the expense of higher communication rate.

Suppose that only one round of interaction from the decoder to the encoder is permitted. In this case, the encoder divides $\mathbf{X}$ into small intervals and sends center-bits, hashes and VT syndromes for each of these intervals. If the intervals are sufficiently small, most intervals will have at most one insertion or deletion and can be corrected using their VT syndromes (and checked using the hashes). The decoder then indicates the intervals it cannot synchronize yet, and the encoder sends these yet-to-be-synchronized intervals in full. The performance of this one-round protocol will be presented in an extended version of this paper.

An important synchronization model that occurs in practice is one where the edits occur in a small number of bursts, but the burst lengths are large ($\Theta(n)$). It is part of ongoing work to extend our protocol to efficiently synchronize in such a situation. Finally, in applications where synchronization to within a targeted Hamming distortion between $\mathbf{X}$ and $\mathbf{Y}$ is required, we can use a distance-aware hash in the protocol, such as the one used in VSYNC [12]. For such a hash, hash-collision is more likely when the sequences being compared are close to one another in Hamming distance. A distance-aware hash is also suitable when there is an outer error-correcting code to correct any residual errors left by the synchronization protocol. Determining the communication rate vs. distortion trade-off in this setting is an interesting direction for future work.

APPENDIX

PROOF SKETCH OF THEOREM 1

Proof of (a) (Protocol Error): The only source of protocol-error is a set of center bits being aligned in the wrong position. (A failure to find a match for a set of center-bits will not directly cause an error since the protocol requests an additional set of center-bits whenever this happens.)

Each time we use $l = c\log_2 n$ center-bits, there is a probability $\frac{1}{n^c}$ of mismatch. Using the union bound, the probability of error is

$$P_e(d) = \frac{\text{Number of times center-bits are compared}}{n^c}. \quad (6)$$

At any stage of the algorithm, there are at most $\frac{d}{2}$ substrings in the unresolved list and an upper bound for the number of rounds is $\log_2 n$. In every round, a set of center-bits is used for each string in the unresolved list. The probability of error is hence at most $\frac{0.5\ d\log_2 n}{n^c}$.

Proof of (b) (Average rate): For each piece under consideration, the encoder sends $l = c\log_2 n$ center-bits using which the decoders determines the number of deletions in the left and right half of the piece. The expected number of bits transmitted

in each direction can be calculated as

$$EN_{1\to 2}(d) = EN_c(d) + d\log_2 n,$$
$$EN_{2\to 1}(d) = 4\ \frac{EN_c(d)}{l} \tag{7}$$

where $N_c(d)$ is the *total* number of center-bits sent (from encoder to decoder) until the protocol terminates, when the number of deletions is d. The first term in $N_{1\to 2}$ above corresponds to the number of center-bits sent. The second term corresponds to the VT syndromes. $N_{2\to 1}$ is obtained by recognizing that the decoder has to signal one of four options for each half of the piece under consideration: 1) Continue splitting, 2) Send VT syndrome, 3) Synchronized or 4) Send additional center bits (i.e., no match found). This signaling takes $2.\log_2 4 = 4$ bits for each piece for which center-bits are sent.

We now obtain a formula for expected number of center-bits $EN_c(d)$. First note that $EN_c(1) = 0$ since we can directly use VT syndromes. When $d = 2$, we are done after one split if the deletions are in different halves, and need to continue splitting if they are in the same half. Hence

$$EN_c(2) = l + \frac{1}{2}EN_c(2) + \frac{1}{2}\cdot 0 \ \Rightarrow\ EN_c(2) = 2l. \tag{8}$$

In general, we have

$$EN_c(d) = l + \sum_{j=0}^{d} \frac{1}{2^d}\binom{d}{j}\left(EN_c(j) + EN_c(d-j)\right)$$
$$= l + \frac{1}{2^{d-1}}EN_c(d) + \sum_{i=1}^{d-1}\frac{1}{2^d}\binom{d}{j}\left(EN_c(j) + EN_c(d-j)\right). \tag{9}$$

Thus we can calculate $EN_c(d)$ using the recursion

$$EN_c(d) = \frac{l + \sum_{i=1}^{d-1}\frac{1}{2^d}\binom{d}{j}\left(EN_c(j) + EN_c(d-j)\right)}{1 - 2^{-(d-1)}} \tag{10}$$

with the initial conditions $EN_c(1) = 0$ and $EN_c(2) = 2l$. Note that we have not considered contribution of the extra center-bits sent when a match is not found. This overhead is $O(\frac{d\log_2 n}{n})$ which goes to zero since d is $o(\frac{n}{\log_2 n})$.

We now show by induction that $EN_c(d) < 2(d-1)l$. The induction hypothesis holds for $k = 2$. Assume it is true for $EN_c(k), k = 1, \ldots, d-1$. Then (10) can be bounded as

$$EN_c(d) < \frac{l + \sum_{i=1}^{d-1}\frac{1}{2^d}\binom{d}{j}\left(2(j-1) + 2(d-j-1)\right)l}{1 - 2^{-(d-1)}}$$
$$= \frac{l + \sum_{i=1}^{d-1}\frac{1}{2^d}\binom{d}{j}(2d-4)l}{1 - 2^{-(d-1)}}$$
$$= \frac{l}{1 - 2^{-(d-1)}} + (2d-4)l < 2(d-1)l. \tag{11}$$

Substituting in (7), we obtain the bounds on the number of bits in each direction.

Proof of (c) (Tail probability of number of rounds): After r rounds of the algorithm, each unresolved piece is at most $2^{-r}n$ bits long, where n is the original length. The algorithm will *not* terminate in r rounds only if there are two deletions spaced less than $2^{-r}n$ positions apart. Since the positions of d deletions are random, for large n, we can normalize by n and think of the deletion positions as d points randomly picked on the unit interval $(0, 1)$.

Hence the protocol terminates within r rounds if *each* of the $d+1$ segments (formed by the d random points) has length at least 2^{-r}. If we denote by $(x_1, x_2, \ldots, x_{d+1})$ the lengths of the segments, the joint distribution $P(x_1, \ldots, x_{d+1})$ is uniform over the unit d-simplex:

$$\{x_i \geq 0\ \forall i,\ \ x_1 + \ldots + x_{d+1} = 1\}. \tag{12}$$

Thus the probability that each segment is at least 2^{-r} long is the probability mass of the subset

$$\{x_i \geq 2^{-r}\ \forall i,\ \ x_1 + \ldots + x_{d+1} = 1\}. \tag{13}$$

By similarity of the two regions in (12) and (13), the desired probability is $(1 - (d+1)2^{-r})^d$.

Thus the probability that the protocol does not terminate after r rounds is $1 - (1 - (d+1)2^{-r})^d$. Substituting $r = k + 2\log_2 d$, and using the Taylor series expansion proves the second part of Theorem 1(c).

REFERENCES

[1] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Doklady Akademii Nauk SSSR*, vol. 163, no. 4, pp. 845–848, 1965. (in Russian), English Translation in *Soviet Physics Dokl.*, (No. 8, 1966), 707-710.

[2] A. Orlitsky and K. Viswanathan, "One-way communication and error-correcting codes," *IEEE Trans. on Inf. Theory*, vol. 49, no. 7, pp. 1781–1788, 2003.

[3] R. R. Varshamov and G. M. Tenengolts, "Codes which correct single asymmetric errors," *Automatica i Telemekhanica*, vol. 26, no. 2, pp. 288–292, 1965. (in Russian), English Translation in *Automation and Remote Control*, (**26**, No. 2, 1965), 286-290.

[4] D. Slepian and J. Wolf, "Noiseless coding of correlated information sources," *IEEE Trans. Inf. Theory*, vol. 19, July 1973.

[5] A. D. Wyner and J. Ziv, "The rate-distortion function for source coding with side information at the decoder," *IEEE Trans on Inf. Theory*, vol. 22, no. 1, pp. 1–10, 1976.

[6] A. Wyner, "Recent results in the Shannon theory," *IEEE Trans. Inf. Theory*, vol. 20, pp. 2–10, Jan 1974.

[7] S. S. Pradhan and K. Ramchandran, "Distributed source coding using syndromes (DISCUS): design and construction," *IEEE Trans. Inf. Theory*, vol. 49, no. 3, pp. 626–643, 2003.

[8] M. Mitzenmacher, "A survey of results for deletion channels and related synchronization channels," *Probability Surveys*, vol. 6, pp. 1–33, 2009.

[9] M. C. Davey and D. J. C. MacKay, "Reliable communication over channels with insertions, deletions, and substitutions," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 687–698, 2001.

[10] G. Cormode, M. Paterson, S. C. Sahinalp, and U. Vishkin, "Communication complexity of document exchange," in *Proc. ACM-SIAM Symp. on Discrete Algorithms*, pp. 197–206, 2000.

[11] A. Tridgell and P. Mackerras, "The rsync algorithm." http://rsync.samba.org/, Nov 1998.

[12] H. Zhang, C. Yeo, and K. Ramchandran, "VSYNC: a novel video file synchronization protocol," in *ACM Multimedia*, pp. 757–760, 2008.

[13] N. J. A. Sloane, "On single-deletion-correcting codes," in *Codes and Designs, Ohio State University (Ray-Chaudhuri Festschrift)*, pp. 273–291, 2000.

[14] A. Orlitsky, "Interactive communication of balanced distributions and of correlated files," *SIAM J. Discrete Math.*, vol. 6, no. 4, pp. 548–564, 1993.

[15] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of Comp. and Sys. Sci.*, vol. 18, pp. 143–154, April 1979.

[16] G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion," *IEEE Trans on Inf. Theory*, vol. 30, no. 5, pp. 766–, 1984.