An Interactive Algorithm for Synchronizing from Burst Deletions

Shuyang Jiang, Clayton Schoeny, and Lara Dolecek
Dept. of Electrical and Computer Engineering, UCLA, Los Angeles, California 90095
Email: {shuyangjiang, cschoeny}@ucla.edu, dolecek@ee.ucla.edu

***Abstract*—We study the efficient synchronization between two distant nodes, A and B, that are connected through a lossless bidirectional communication channel. In our setup, Node A contains file X, and node B contains file Y that is generated through burst deletions from X. The burst deletion pattern is modeled by a stationary two-state Markov chain. Inspired by the previously proposed synchronization protocol of Yazdi and Dolecek that is specifically designed to handle i.i.d. deletions, we create a new synchronization algorithm that is efficient for the case of burst deletions. Utilizing the previous overall framework from Yazdi and Dolecek, we first prove that the matching module still works well in our scenario. We then invent two new schemes for the implementation of the burst deletion recovery module. The remaining errors created in the first two modules are eliminated by the final LDPC decoder. Lastly, we provide experimental results of the proposed synchronization algorithm for both i.i.d. and realistic sources.**

I. Introduction

Consider two nodes A and B that are connected through a two-way communication channel. Node A holds file X, and node B holds file Y, which is derived from file X through some deletions. We assume that these deletions appear in bursts, which is more representative of practical scenarios. The locations of deleted bits are unknown to either node. We seek to fully recover file X at node B in an efficient manner. For efficiency, we are concerned with the overall bandwidth, which is measured by the total amount of data exchanged by the two nodes, as well as the number of interactive communication round. Additionally, we desire to reconstruct file X from file Y with a suitably low probability of error. This problem of file synchronization arises in many practical applications, including data storage, file sharing, and cloud applications.

There has already been a large body of research work on synchronization of two remote files. One collection of work has concentrated on synchronizing from a prescribed number of edits between two files X and Y. In [1], Levenshtein built upon the coding scheme of Varshamov and Tenengolts and showed that Varshamov-Tenengolts (VT) codes are capable of correcting a single insertion or deletion edit on binary file strings. Orlitsky established several fundamental bounds on the minimum number of transmitted bits for a prescribed edit distance when interactive communication between two nodes is allowed [2]. The work of Orlitsky was followed by a series of works such as [3], [4] and [5], which give explicit protocols for interactive communication. More recently, Venkataramanan *et al.* [6] developed a low-complexity synchronization scheme, which can correct a sublinear rate of edits with near-optimal communication rate. It uses a divide-and-conquer approach to isolate edits and efficiently split the source sequence into substrings containing exactly one deletion or insertion. Each of these substring is then synchronized using an optimal one-way algorithm based on VT codes. The authors generalized their results in [7], in which the protocol is modified to deal with more general cases such as bursts, substitution errors, and limited rounds of communication.

In recent years, works have focused on the case where the number of edits is proportional to the length of the file. This setting is more typical in real world applications. In [8], a deterministic, polynomial-time synchronization scheme is proposed. For this synchronization protocol, the edited file Y is obtained from the binary uniform source file X by deleting each bit independently with the same small probability β. It has order-optimal communication rate, polynomial computational complexity, and the error probability is exponentially low in the size of X. There are three parts of the protocol: the matching module, the deletion recovery module, and the LDPC decoder. The deletion recovery module essentially comes from the previously introduced algorithm proposed by Venkataramanan *et al.* [6]. Based on [8], several extensions to the synchronization protocol [9]–[11] have been proposed. In a recent work by Ma *et al.* [12], achievability bounds were given on the communication rate for synchronization from deletions. The deletions are viewed as the output of a Markov process, and hence the number of deletions is linear in the file length.

In addition, there exist some practical synchronization tools. One of these utilities is rsync [13], which is very robust and only requires a single round of communication. However, it can be in general very inefficient and the number of transmitted bits can be exponentially larger than the optimal one.

In most previous works that are focused on synchronizing from a fixed rate of edits between two files X and Y, it is assumed that every bit of X is edited independently with the same probability. In this paper, we focus on the synchronization from burst edits, since burst edits are, in general, more realistic than i.i.d. edits. For simplicity, we only consider binary uniform source file X, and limit the edits to short bursts of deletions. And we utilize a stationary two-state Markov chain as our error model.

Research supported in part by NSF grant CCF-CIF-1527130

978-1-5386-7048-4/18/$31.00 ©2018 IEEE

In [12], a Markov chain is used to model the burst deletions. However, [12] does not offer an explicit, deterministic construction for a synchronization protocol. We construct a valid, explicit protocol for synchronizing from short bursts of deletions. We adopt the overall framework from [8]. We prove that the matching module works well in our scenario, and create two new algorithms for the burst deletion recovery module. Furthermore, we evaluate the performance of our protocol in experimental simulations and demonstrate its improved efficiency in the case of both i.i.d. and realistic sources.

II. Problem Setting

A. Problem Statement

We represent a binary file string X of length n by $X = (X_1, X_2, ..., X_n)$, in which $X_i \in \{0, 1\}$. The deletion channel is a channel that can delete any subset of the bits of the input file string. We let X and Y be the input and the output of the deletion channel, respectively. The set of deleted bits from the input file string is represented by a binary vector $D = (D_1, D_2, ..., D_n)$. We call D the deletion pattern. If X_i is deleted from X, we have that $D_i = 1$ and otherwise $D_i = 0$. In this paper, we consider the case of burst deletion patterns, i.e., consecutive 1's in D. For the burst deletion pattern D, we define a function f_D, which maps the indices of bits in X to their corresponding indices in Y. More specifically, for index i, if $D_i = 0$, then $f_D(i) = i - \sum_{j<i} D_j$, and if $D_i = 1$, then $f_D(i) = f_D(i')$, where i' is the largest index smaller than i for which $D_{i'} = 0$.

Then, the problem of file synchronization from burst deletions that we need to address is as follows: Let node A and B contain binary file strings X and Y, respectively. Y is the output of a deletion channel with input X and burst deletion pattern D. The burst deletion pattern is unknown to both nodes A and B. We suppose that the source file string X is generated by an i.i.d. Bernoulli source with parameter $\frac{1}{2}$. The burst deletion pattern is mathematically modeled as a stationary two-state Markov chain, which will be discussed in detail in Section II-B. We are interested in a synchronization algorithm on a two-way channel between nodes A and B so that node B can recover file string X from Y with a small probability of error.

B. The Burst Deletion Pattern

In order to model the burst deletion pattern $D = (D_1, D_2, ..., D_n)$, we use a stationary two-state Markov chain. The transition probabilities $P(D_i = 0|D_{i-1} = 0) = 1 - P(D_i = 1|D_{i-1} = 0) = p_1$, and $P(D_i = 1|D_{i-1} = 1) = 1 - P(D_i = 0|D_{i-1} = 1) = p_2$, for all $i = 2, 3, ..., n$. D_1 follows the stationary distribution of the Markov chain, i.e., $P(D_1 = 1) = 1 - P(D_1 = 0) = (1 - p_1)/(2 - p_1 - p_2)$. The schematic diagram of this Markov chain model is shown in Fig. 1. From the definitions of p_1 and p_2, we know that both p_1 and p_2 should be very close to 1 in order to make the deleted bits appear in bursts. Next, we will explore properties of the proposed Markov chain model for the burst deletion pattern.

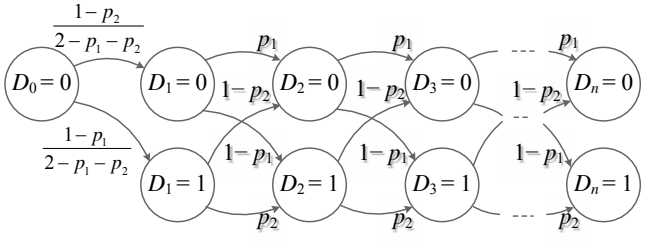

Fig. 1. The proposed model for generating burst deletion pattern D. $D_i = 1$ means that the ith bit is deleted, and $D_i = 0$ means that it is not deleted.

Property 1: For the burst deletion pattern generated by the proposed Markov chain model, the burst length of non-deleted bits and deleted bits, B_{nd} and B_d, follow geometric distributions: $B_{nd} \sim Geo(1 - p_1)$ and $B_d \sim Geo(1 - p_2)$. Hence, the average burst length of non-deleted bits and deleted bits are $\overline{B_{nd}} = \frac{1}{1-p_1}$ and $\overline{B_d} = \frac{1}{1-p_2}$, respectively, and the average deletion rate is $\overline{d} = \frac{1-p_1}{2-p_1-p_2}$.

Property 2: For the burst deletion pattern D of length $n = 1/\overline{d}$ ($\overline{d} \ll 1$) generated by the proposed Markov chain model, we denote the number of deletion bursts within D as Δ_n. We then have

$$P(\Delta_n = 0) \geq p_2(1 - \overline{d}),$$
$$P(\Delta_n > 1) \leq \overline{d}(1 - p_2) + (1 - p_2)^2.$$

From Property 2, we know that within a burst deletion pattern of length $1/\overline{d}$, there is a very high probability to have no deletion bursts and a very low probability to have more than 1 deletion bursts. This property will be helpful for the design of the burst deletion recovery module.

C. Assumptions

To make our file synchronization problem more tractable, we make two simplifying assumptions for the burst deletion pattern:

1) The length of most deletion bursts B_{nd} should be no larger than a predefined maximum length $B_{\max}$ (e.g., $B_{\max} = 10$),
2) The average deletion rate $\overline{d}$ is quite small (e.g., $\overline{d} = 0.005, 0.01$).

In order to satisfy assumption 1), we have $P(B_{nd} \leq B_{\max}) = (1-p_2)(1+p_2+...+p_2^{B_{\max}-1}) = 1-p_2^{B_{\max}}$, which should be close to 1. Assumption 2) usually holds like in [8] and [9]. For this assumption, we get that $\overline{d} = \frac{1-p_1}{1-p_1+1-p_2} \ll 1$, i.e., $1 - p_1 \ll 1 - p_2$. In addition, p_2 should be close to 1 so that the deleted bits appear in bursts. Hence, the requirements for the parameters p_1 and p_2 of the proposed Markov chain model are as follows:

1) $p_2^{B_{\max}} \ll 1$, but p_2 is close to 1,
2) For every fixed p_2, we have various p_1 that satisfies that $1 - p_1 \ll 1 - p_2$.

III. Synchronization Protocol

In this section, we propose a new synchronization protocol, which can correct short bursts of deletions in an efficient way.

We build our synchronization algorithm based on the work of [8], and use the same framework with it. Similarly, the new protocol also consists of three modules: the matching module, the burst deletion recovery module, and the LDPC decoder module. The matching module is adapted to our scenario in a very straightforward way, while the burst deletion recovery module is redesigned by using new algorithms.

A. Protocol Overview

Since our protocol has the same framework with that of [8], one can find that the main processes of two protocols are quite similar. However, the detailed implementations of first two modules, especially the burst deletion recovery module, are quite different. We first take a brief overview of the proposed protocol.

To begin, node A partitions the file X into segment substring S_i ($1 \leq i \leq k$) and pivot substring P_i ($1 \leq i \leq k-1$). Then, file X becomes as follows:

$$X = S_1, P_1, S_2, P_2, ..., S_{k-1}, P_{k-1}, S_k,$$

in which $|S_i| = L_S = 1/\overline{d}$ and $|P_i| = L_P = O(\log(1/\overline{d}))$. $\overline{d} = \frac{1-p_1}{2-p_1-p_2}$ is the average deletion rate. Note that the length of a pivot string L_P is much smaller than the length of a segment string L_S. Based on Property 2, We choose the length of the segment string $L_S = 1/\overline{d}$ so that most of segment strings has 0 or 1 deletion burst.

Then, node A sends pivot strings $P_1, P_2, ..., P_{k-1}$ to node B in order. And the matching module attemps to find the correct copies of these pivots within file Y. Because of the deletions in the file, the matching module is only able to find the matches for a subsets of P_i's, which are denoted as $P_{i_1}, ..., P_{i_{k'-1}}$, $1 \leq i_1 < i_2 < ... < i_{k'-1} \leq k-1$, where $k' \leq k$. Based on the positions of matched P_i's, the matching module divides file Y into substrings as

$$Y = \overline{F}_1, P_{i_1}, \overline{F}_2, P_{i_2}, ..., \overline{F}_{k'-1}, P_{i_{k'-1}}, \overline{F}_{k'}.$$

The indices of matched pivots $\{i_1, ..., i_{k'-1}\}$ are then sent back to node A, which accordingly divides file X into

$$X = F_1, P_{i_1}, F_2, P_{i_2}, ..., F_{k'-1}, P_{i_{k'-1}}, F_{k'},$$

So far, we have gotten many pairs of segment strings $\{(\overline{F}_j, F_j), j = 1, 2, ..., k'\}$. As will be explained in Section III-C, $\overline{F}_j$ can be derived from F_j by very few (mostly 0 or 1) short bursts of deletions if both $P_{i_{j-1}}$ and P_{i_j} are matched correctly in Y. Then, we need to synchronize $\overline{F}_j$ with the corresponding F_j, which is finished by the burst deletion recovery module. Since deletions appear in bursts, it is no longer efficient to use the divide-and-conquer approach to isolate deletions until each matched substring only contains one deletion. In our burst deletion recovery module, two different new schemes are used. One scheme still uses the divide-and-conquer approach while the other one does not. Compared with the first scheme, the second one needs far less (at most two) rounds of interaction but consumes more bandwidth. It is verified that these two new algorithms work well in our case. At the end of this step, we recover the estimates of all F_j's from $\overline{F}_j$'s. And an estimate of the original file X is gotten as:

$$\tilde{X} = \tilde{F}_1, P_{i_1}, \tilde{F}_2, P_{i_2}, ..., \tilde{F}_{k'-1}, P_{i_{k'-1}}, \tilde{F}_{k'},$$

which has the same length as file X. At the last step, we use the LDPC decoder module to correct the residual errors created from the erroneous match of pivot strings and the algorithm used in the burst deletion recovery module.

Up to now, we have taken an overview of the new synchronization protocol. Next, we recall a theorem from [12], which provides the minimum rate of any synchronization algorithm under the burst deletion channel modeled by Markov chain.

Theorem 1 (Ma et al., [12]): We consider the synchronization algorithm under the burst deletion channel, which is modeled by the stationary Markov chain. If $1-p_1 \ll 1$ and p_2 is fixed, for any $\epsilon > 0$, we have the minimum communation rate

$$R_{\min}(p_1, p_2) = -(1-p_1)\log(1-p_1) + (1-p_1)\left(\frac{1+h_2(1-p_2)}{1-p_2} + \log e - C\right) + O((1-p_1)^{2-\epsilon}),$$

where $C = \sum_{l=1}^{\infty} 2^{-l-1} l \log l \approx 1.29$, and $h_2(x) = -x\log x - (1-x)\log(1-x)$.

The above theorem considers the case when $1 - p_1 \ll 1$ and p_2 is fixed, which coincides with our assumptions given in section II-C. From this theorem, we can get the lower bound of communication rate for our new synchronization protocol.

In the following sections, we give further details into the matching module and the burst deletion recovery module.

B. Matching Module

The task of the matching module is to detect correct matches of P_i's within file Y. In [8], a graph theoretic method is used to construct this module. The method makes use of a matching graph to get the matches of P_i's. Adapting the matching module from the case of i.i.d. deletion pattern in [8] to our scenario of burst deletion pattern is quite straightforward. The construction of the matching graph and some relevant mathematical analysis can be made in a very similar fashion. Before introducing the construction of the matching module, we need to formalize the notion of correct and incorrect matches for P_i's.

For any pivot P_i, there might be many matches for it within file Y. Because of this, we need to define what is the correct match of P_i. Let us denote the index of the first bit of P_i in X by $\check{p}_i$ and the index of the last bit of P_i in X by $\hat{p}_i$. Based on the number of deletions that acts on each P_i in X, we have the following two cases:

1) There is no deletion within P_i. Then, the copy of P_i in Y between indices $f_D(\check{p}_i)$ and $f_D(\hat{p}_i)$ is the correct match while all other copies of P_i in Y are incorrect matches.
2) There is at least one deletion within P_i. Then, all copies of P_i in Y are incorrect matches.

In contrast with [8], we do not consider the case where exactly one deletion occurs in P_i, since the probability of one

deletion in P_i is much smaller in the burst deletion pattern case compared with the i.i.d. deletion pattern case.

Similar to the theoretical analysis about the occurrence of correct matches for P_i in [8], we have the following conclusion:

Remark 1: With probability $(1-\overline{d})p_1^{L_P-1} \geq 1-L_P\overline{d}+o(\overline{d})$, P_i has no deletion and there is one correct match for P_i within Y. And with probability $1-(1-\overline{d})p_1^{L_P-1} \leq L_P\overline{d}+o(\overline{d})$, P_i has at least one deletion and there is no correct match for P_i within Y.

Similarly, we define $R = 1 - L_P\overline{d}$. From Remark 1, we conclude that

Remark 2: For a random string X and a random burst deletion pattern D defined in Section II, on average, the number of pivots with one correct match in Y is $(1-\overline{d})p_1^{L_P-1}k$, which is no less than $(R+o(\overline{d}))k$.

Remark 3: For a random string X and a random burst deletion pattern D defined in Section II, with probability $1-2^{-\Omega(n)}$, there are $((1-\overline{d})p_1^{L_P-1}+o(\overline{d}))k \geq (R+o(\overline{d}))k$ pivots with one correct match in Y.

The proof of the above three remarks is quite similar to that of the corresponding results in [8]. For details, please refer to [8]. From Remark 3, we see that most of pivots has one correct match in Y in the case of burst deletion pattern.

Note that the theoretical results about the occurence of correct matches of P_i in the previous discussion are quite simiar with those in [8]. Both our protocol and the protocol of [8] focus on the deletion-only edits. This inspires us to use the same construction of the matching module with [8] (i.e., the same matching graph) to find the correct matches of P_i. The detailed implementation of the matching graph can be found in [8]. In addition, we make two slight modifications for the matching graph as follows:

1) For any two vertices in layers i and j with $i < j$, we connect these two vertices if and only if the distance between them is within $[0, (j-i-1)L_P+(j-i)L_S]$ (In [8], the distance is within $[-1, (j-i-1)L_P+(j-i)L_S]$). This is due to the small different definition of correct and incorrect matches for P_i.
2) We need to find a path of length $(1-\overline{d})p_1^{L_P-1}k+o(\overline{d})k$ from the begining vertex s to the ending vertex t in the matching graph (In [8], this length is $(1-L_P\beta+2\beta)k+o(\beta)k$ where β is the deletion rate).

The numerical experiment shows that, for a proper L_P, it is with a very high probability that nearly all detected matches of P_i's are correct matches. However, to get a theoretical guarantee of this like that in [8], is rather difficult and is left for future investigation.

C. Burst Deletion Recovery Module

After the matching of pivots, we have divided the problem of synchronizing Y to X into many subproblems. For each subproblem, we need to synchronize a segment string $\overline{F}_j$ in Y to its orignial one F_j in X, which is the goal of the burst deletion recovery module. Since most of the pivots can be correctly matched, we expect that most of segment strings F_j's in X are of length $1/\overline{d}$. From Property 2, we know that most of $\overline{F}_j$'s in Y are derived from the corresponding F_j's in X by 0 or 1 short burst of deletions. This observation inspires us to develop a new algorithm for the burst deletion recovery module as follows.

In the first round, if the lengths of $\overline{F}_j$ and F_j differ by 0, 1, or more than 1, we would declare $\overline{F}_j$ synchronized, send a VT syndrome, or use a single-deletion-burst synchronization algorithm, respectively. Then after the first round, most of $\overline{F}_j$'s are succussfully synchronized. Only a small percentage of $\overline{F}_j$'s, which are derived by multiple short deletion bursts, need additional rounds of interaction to synchronize. For these $\overline{F}_j$'s in Y, we use the divide-and-conquer approach to split them into smaller segment strings until each of segment strings differs by 0 or 1 short deletion burst from the corresponding segment string in X.

In our new algorithm, one key submodule is the single-deletion-burst synchronization algorithm. In order to synchronize from a single deletion burst of length l_B, we use the one-way protocol introduced in [6]. The segment string x of length n in X is divided into l_B substrings so that every substring contains only one deletion. Then by sending the VT syndromes of these l_B substrings to node B, x can be recovered at node B. The bandwidth for this protocol is $l_B \log(n/l_B+1)$ bits. In [7], an improved single-deletion-burst synchronization algorithm is derived, which only costs at most $4\log(1+n/l_B)+3(l_B-2)$ on average. However, compared with the one-way protocol in [6], this improved algorithm needs an additional round of interaction and has an comparable bandwidth when l_B is small. Hence, we do not adopt this improved algorithm in our protocol.

We summarize our new algorithm as follows:

1) In the first round, if the lengths of $\overline{F}_j$ and F_j differ by 0, 1 or $l_B > 1$, we would declare $\overline{F}_j$ synchronized, send a VT syndrome, or send l_B VT syndromes with hash, respectively. For those pairs of matching segment strings where hashes do not agree after l_B VT decoding, we put them in the unresolved list $\mathcal{L}_X$ at node A and the corresponding list $\mathcal{L}_Y$ at node B.
2) In each round, node A sends m_a anchor bits around the center of each substring in $\mathcal{L}_X$ to node B. Node B tries to align these bits as close as possible to the center of the corresponding substring in $\mathcal{L}_Y$. If a match is found, the substring is split into two pieces $str1$ and $str2$.
   - If one piece $str1$ contains no deletion, we declare $str1$ synchronized. For the other piece $str2$, node B puts this piece in $\mathcal{L}_Y$ (and instructs node A to put its corresponding piece in $\mathcal{L}_X$).
   - If both $str1$ and $str2$ contain some deletions, we perform the following steps for each piece: 1) if the number of deletions is 1, node B requests the VT syndrome for synchronization; 2) if the number of deletions is $l_B > 1$, node B requests l_B VT syndromes with hash. If hashes match after l_B VT decoding, we declare the piece synchronized; oth-

erwise, node B puts this piece in $\mathcal{L}_Y$ (and instructs node A to put its corresponding piece in $\mathcal{L}_X$).

If node B can not align the anchor bits due to deletions, it requests another set of m_a anchor bits for the substring, which is chosen adjacent to a previously sent set of anchor bits and as close to the center of the substring as possible.

3) The process continues until $\mathcal{L}_Y$ (or $\mathcal{L}_X$) is empty.

In addition, whenever a yet-to-be-synchronized substring is short enough that sending l_B VT syndromes with hash or sending anchor bits is not meaningful, the substring is just sent in full.

Based on the just introduced new algorithm, another simplified algorithm without the divide-and-conquer approach can be derived as follows:

1) The first step is the same with the above algorithm.
2) For all substrings in $\mathcal{L}_X$, node A send these substrings in full to node B and hence complete the synchronization.

In the second step of simplified algorithm, all substrings in $\mathcal{L}_X$ contain more than 1 deletion bursts and account for a very small percentage of all F_j's. Hence, sending all these substrings in full would not consume much more bandwidth, while the round of interaction is reduced to at most two rounds.

IV. Experimental Results

We compare the bandwidth of our synchronization protocols with the bound of Theorem 1. The number of rounds used by our protocols is also recorded. In the experiments, we first use the i.i.d. Bernoulli(0.5) binary sequence of length $n = 100k$ as source file X. File Y is derived from file X by applying our proposed burst deletion pattern with parameters $(\overline{d}, p_2) = (0.005, 0.82)$, $(0.002, 0.82)$ and $(0.005, 0.91)$ (p_1 is hence determined). Then, we consider some realistic data by using a Mona Lisa image and Martin Luther King's speech text as source file. The image and text are first converted to binary strings of length 231200 and 63623 respectively, and then fed into a burst deletion channel with parameters $(\overline{d}, p_2) = (0.005, 0.82)$ to get file Y. For the matching module, we have that $L_S = 1/\overline{d}$ and $L_P = 10$. We further set the length of anchor bits $m_a = 6$ and the hash length as 5 bits. The results are averaged over 100 trials for i.i.d. source file and over 1 trial for realistic data. Table I shows the results.

As shown in the table, when file X is i.i.d. binary string, the expected bandwidth of both protocols is within a small factor (around 6 in our simulations) of the bound in [12]. The simplified protocol requires only 3 rounds of interaction in total but slightly more bandwidth compared with the original new protocol. When we increase p_2 from 0.82 to 0.91 (i.e., the deletion bursts have larger length on average), the simplified protocol needs nearly the same bandwidth with original one. This is because that much fewer segment substrings contain more than 1 deletion bursts and sending these substrings in full adds very little bandwidth. In the case of realistic data sets, our protocols are still very efficient with their bandwidth larger than the lower bound by a small factor (about 6), and need moderate rounds of interaction.

TABLE I
The average bandwidth ($\overline{B}$) and average interactive rounds ($\overline{R}$) used by the protocols of ours, with lower bound of bandwidth $B_{\min}$ given in Theorem 1 [12].

$(\overline{d}, p_2)$	Our protocol		Simplified protocol		$B_{\min}$ [12]
	$\overline{B}$	$\overline{R}$	$\overline{B}$	$\overline{R}$	
(0.005, 0.82)	9.23k	13.79	10.32k	3	1.77k
(0.002, 0.82)	4.05k	11.70	5.21k	3	0.75k
(0.005, 0.91)	8.13k	11.79	8.30k	3	1.23k
King's speech text (0.005, 0.82)	6.47k	10	7.31k	3	1.13k
Mona Lisa image (0.005, 0.82)	21.39k	13	24.35k	3	4.10k

V. Conclusion

We studied the problem of file synchronization under burst deletions. To generate the burst deletion pattern, a stationary Markov chain is used. Based on the work of [8], we adapt the matching module and redesign the burst deletion recovery module. Two new protocols are derived, which have improved efficiency for both i.i.d. and realistic sources. Further work involves the theoretical analysis of total bandwidth and the extension to the case of both burst deletions and insertions.

References

[1] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals (in Russian)," *Soviet Phys. Doklady*, vol. 163, no. 4, pp. 845-848, 1965.

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

[3] G. Cormode, M. Paterson, S. C. Sahinalp, and U. Vishkin, "Communication complexity of document exchange," in *Proc. 11th Annu. ACMSIAM Symp. Discrete Algorithms*, San Francisco, CA, USA, Jan. 2000, pp. 197-206.

[4] A. V. Evfimievski, "A probabilistic algorithm for updating files over a communication link," in *Proc. 9th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA)*, San Francisco, CA, USA, Jan. 1998, pp. 300-305.

[5] A. Orlitsky and K. Viswanathan, "Practical protocols for interactive communication," in *Proc. IEEE Int. Symp. Inf. Theory*, Washington, DC, USA, Jun. 2001, p. 115.

[6] R. Venkataramanan, H. Zhang, and K. Ramchandran, "Interactive low complexity codes for synchronization from deletions and insertions," in *Proc. IEEE 48th Allerton Conf. Commun., Control, Comput.*, Monticello, IL, USA, Sep./Oct. 2010, pp. 1412-1419.

[7] R. Venkataramanan, V. N. Swamy, and K. Ramchandran, "Low-complexity interactive algorithms for synchronization from deletions, insertions and substitutions," *IEEE Trans. Inf. Theory*, vol. 61, pp. 5670-5689, Oct. 2015.

[8] S. M. S. Tabatabaei Yazdi, and L. Dolecek, "A deterministic polynomial-time protocol for synchronizing from deletions," *IEEE Trans. Inf. Theory*, vol. 60, no. 1, pp. 397-409, Jan. 2014.

[9] F. Sala, C. Schoeny, N. Bitouz, and L. Dolecek, "Synchronizing files from a large number of insertions and deletions," *IEEE Trans. Communications*, vol. 64, pp. 2258-2273, Jun. 2016.

[10] C. Schoeny, N. Bitouz, F. Sala, and L. Dolecek, "Efficient file synchronization: extensions and simulations," in *Signals Systems and Computers 2014 48th Asilomar Conference on*, pp. 2129-2133, 2014.

[11] N. Bitouz, F. Sala, S. M. S. Tabatabaei Yazdi, and L. Dolecek, "A practical framework for efficient file synchronization," in *Proc. 51st Annu. Allerton Conf. Commun., Control, Comput.*, pp. 1213-1220, Oct. 2013.

[12] N. Ma, K. Ramchandran, and D. Tse, "Efficient file synchronization: a distributed source coding approach," in *Proc. IEEE Int. Symp. Inf. Theory*, pp. 583-587, 2011-Jul./Aug.

[13] A. Tridgell, "Efficient algorithms for sorting and synchronization," Ph.D. dissertation, Dept. Comput. Sci., Austral. Nat. Univ., Canberra, Australia, 2000.