UNIVERSITY OF CALIFORNIA
Los Angeles

An Interactive Algorithm for Synchronizing From Burst Deletions

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Electrical Engineering

by

Shuyang Jiang

2018

ABSTRACT OF THE THESIS

An Interactive Algorithm for Synchronizing
From Burst Deletions

by

Shuyang Jiang
Master of Science in Electrical Engineering
University of California, Los Angeles, 2018
Professor Lara Dolecek, Chair

We consider the synchronization between two distant nodes A and B that are connected through a two-way communication channel. Node A contains file X, and node B contains file Y that is generated through i.i.d. deletions from X. In previous work, a deterministic polynomial-time protocol for reconstructing file X at node B is proposed, which has the order-wise optimal rate and exponentially low probability of error.

In this thesis, we consider the case of burst deletions, which is more applicable in practical scenario compared with i.i.d. deletions. In order to model this new deletion pattern, we use a stationary two-state Markov chain. Based on previous protocol, we offer a new synchronization scheme specifically designed for burst deletion pattern. The experimental result shows that our proposed protocol works well.

The thesis of Shuyang Jiang is approved.

Jonathan Chau-Yan Kao

Tyson Condie

Lara Dolecek, Committee Chair

University of California, Los Angeles

2018

To my father and mother

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Developing efficient algorithms to synchronize between different versions of files is a meaningful problem with numerous applications. We consider a synchronization problem between two distant nodes A and B that are connected through a two-way, error-free communication channel. Node A contains a binary file X of length n, and node B contains a binary file Y that is an edited version of X. The edits may consist of deletions, insertions and substitutions of bits. Usually, the locations of edited bits are *unknown* to both nodes. We seek to reconstruct file X at node B in an efficient way. For efficiency, we are mainly concerned with the number of transmitted bits between the two nodes and the complexity of implementing the protocol at nodes A and B. Also, as usual, we desire to reconstruct file X at node B with the error probability that is vanishing when the length of file goes into infinity.

For simplicity, we limit the edits to deletions in the thesis. Below is an example of how the file Y can be derived from file X by some deletions:

$$X = 00\underset{D}{1}01\underset{D}{1}00\underset{D}{0}\underset{D}{1}0101\underset{D}{1}1, \text{ and}$$
$$Y = 00010001011,$$

where deleted bits are denoted by D. We can see that the file Y is obtained from the original file X by five deletions. The function of file synchronization is to recover the file X at node B based on the file Y with the minimal amount of exchange information between node A and node B. It is well known that synchronizing file strings can be viewed as a special case of the general problem of *object reconciliation* [1].

1.1 Motivation

File synchronization is the subject of many practical applications, including data storage, file sharing, and cloud applications. For example, as the popularity of cloud computing grows, many technology companies, such as Google and Dropbox, are required to maintain exponentially growing data-storage systems. Because hard drives are prone to failure, these companies must store many back-up copies of a file. Whenever files are changed, synchronization between the changed file and the back-up copies should be performed. Also, synchronization tools are necessary in daily devices. Changes made to a pdf file on a laptop need to be synchronized with the corresponding one on a tablet or smart phone. Furthermore, synchronization tools are useful for video and sound editing, data deduplication, and DNA sequencing.

A naive approach for synchronization is to simply transmit the entire file X from node A to node B. However, the two files to be synchronized are usually very similar with inherent redundancy. So the transmission of the whole file is very inefficient. A simple lower bound of communication required to synchronize from the edits can be obtained by assuming that node A knows the locations of the edits in file X. Then, the minimum exchange information is the contents and positions of the edited bits. Hence, how to design an efficient synchronization algorithm with the communication bandwidth close to this lower bound is very significant.

1.2 Existing Work

There has already been a large body of research work on synchronization of two remote files. One kind of previous work has concentrated on synchronizing from a prescribed number of edits between two files X and Y. In [2], Varshamov and Tenengolts presented a coding scheme, which can recover from one asymmetric error. Later in [3], 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. It is known that the family of VT codes with checksum parameter $a = 0$ have asymptotically

optimal rate (as the code length goes to infinity) and are conjectured to be optimal in all cases. Tenengolts [4] altered the construction of the VT code to allow for the correction of a single insertion or deletion on a non-binary string. Constructions of codes, which are capable of correcting multiple insertion and deletions, are given in [5] and [6]. However, such codes have a low rate. In the case of repetion errors, Dolecek [7] presents a code construction based on a generalization of the VT codes, which is rate optimal.

All of the previously described works are focused on the one-way communication setting. Instead of performing a one-directional transfer of information, the interaction between two nodes is introduced for the synchronization algorithms that can correct a fixed amount of edits. The interaction allows the synchronization to be more efficient in terms of the bandwidth at the cost of multiple rounds of communication. In [8], Orlitsky established several fundamental bounds on the minimum number of transmitted bits under a permitted number of communication rounds for a prescribed edit distance. This work was followed by a series of works such as [9], [10] and [11], which give explicit protocols for interactive communication. For the original file of length n with δ number of edits, Cormode [9] offered an ϵ-error algorithm with $c(\epsilon)\delta \log^3 n$ total transmitted bits [1], in which $c(\epsilon)$ is a constant dependent on ϵ. In [10], Evfimievski gave out a protocol with the number of transimitted bits polynomial in $\log n$, $\log 1/\epsilon$, and δ. When the number of edits is unknown, Orlitsky [11] showed that it needs at most $\delta \log n + \log 1/\epsilon$ transimitted bits for the ϵ-error optimal protocol. An explicit synchronization protocol is also given out in [11], which needs $2\delta \log n(\log n + \log\log n + \log 1/\epsilon + \log \delta)$ transmitted bits. Venkataramanan [12] developed a low-complexity synchronization scheme, which can correct $\delta = o(n/\log n)$ 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 single-deletion correcting codes introduced by Varshamov and Tenengolts [2]. It needs $(4c+1)\delta \log n$ transmitted bits from node A to node B and $10(\delta - 1)$ transmitted bits from node B to node A for given positive number c. The authors generalized their results

[1] All logarithms in this thesis are in base 2.

in [13], in which the protocol is modified to deal with more general cases such as bursts, substitution errors and limited rounds of communication.

More recently, there appears another branch of synchronization algorithms, which focus 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 [14], 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 in the protocol: the matching module, the deletion recovery module and the LDPC decoder, in which a key component module comes from the previously introduced algorithm proposed by R. Venkataramanan [12]. Based on [14], several extensions to the synchronization protocol have been proposed. In [15], F. Sala extended the protocol in [14] in three ways. The files can be nonbinary and nonuniform instead of being binary uniform. And the edits can be both insertions and deletions. C. Schoeny [16] introduced an adaptive algebraic code to the original protocol, in order to increase the effciency in the presence of substitution errors. Furthermore, the performace of the protocol is evaluated in [17], and significant gains over a popular Unix utility rsync are reported. In a recent work by Ma [18], 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, which are not based on coding-theoretic ideas. One of such protocols is rsync [19]. This protocol is a UNIX-based synchronization tool based on an algorithm which uses two hashes, one strong and one weak. The edited file is split into segments of fixed length, and the rolling hash is applied to each such segment as well as all consecutive segments in the original file. These hashes are compared in order to match segments in the original and edited files. Matching segments are checked for equality using the strong hash. If this hash fails, the entire segment is sent to the edited version of the file. rsync is very robust and only requires one 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. There are also some more specialized synchronization tools, such as vsync [20], which is for the synchronization of video files.

1.3 Our Contribution

For some previous work that is focused on synchronizing from a fixed rate of edits between two files X and Y, they assume that every bits of X is edited independently with the same probability. However, in practical scenario, this i.i.d. pattern of edited bits is not applicable. Burst edits can be a major source of mis-synchronization, as we often edit chunks of a file rather than isolated bits. In this thesis, we focus on the file synchronization in the case of burst edits. For simplicity, We let the source file X be binary and uniformly distributed, and limit the edits to deletions. Possible extensions to the more general case of both burst deletions and insertions will be discussed at the end of the thesis. In order to model the burst deletions, we utilize a stationary two-state Markov chain. We will also discuss some properties of this Markov model in detail, which shows that the proposed Markov chain is suitable to model the burst deletions. Based on the given properties, we make some assumptions to simplify our synchronization problem.

In [18], the Markov chain has already been used to model the burst deletions. However, it does not offer any explicit, deterministic construction for the synchronization protocol. We try to construct a valid, explicit protocol for synchronizing from burst deletions. Since the burst deletion pattern destroys the divide-and-conquer approach of isolating deletions used in [14], this protocol is not efficient any more. However, the framework of the protocol in [14] is still inspiring for our work. We build our synchronization protocol based on the work of [14], and the numerical results demonstrate that our proposed protocol works well with a small communcation rate and lower number of interactive rounds.

The rest of the thesis is organized as follows. In Chapter 2, we present the problem setting and discuss some useful properties of the proposed Markov chain model. In order to simplify the synchronization problem in burst deletion setting, some assumptions are made.

In Chapter 3, we give out a review of the protocol in [14] by introducing the main purpose of each module. In addition, we delve deeper into specific implementation details of the protocol and some relevant mathematics behind the algorithm. Then, in Chapter 4, our new synchronization protocol, which is designed for the case of burst deletions modeled by the Markov chain, is presented. Chapter 5 contains some experiment results, which demonstrate that our protocol works well. And the conclusion and some furture work is given in Chapter 6.

CHAPTER 2

Problem Setting

In this chapter, we formally state our problem for the synchronization from burst deletions. Besides, in order to mathematically model the burst deletions, a stationary two-state Markov chain is used. We also give out several useful properties of this mathematical model. From these properties, we can see that the stationary Markov chain is a proper model to generate the burst deletions. To highlight the main ideas and keep the exposition simple, we focus on the case where the file that need to be synchronized is a binary sequence. The extension to larger discrete alphabets is very straightforward.

2.1 Problem Statement

We first introduce some notations that need to be used throughout the thesis. We represent a binary file string X of length n by $X = (X_1, X_2, ..., X_n)$, in which $X_i \in \{0, 1\}$. For $1 \leq i \leq j \leq n$, $X(i, j)$ is the substring $(X_i, X_{i+1}, ..., X_j)$ of X. For a file string X, we let $|X|$ denote the length of X.

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 $D = (D_1, D_2, ..., D_n)$ of length $|X|$. 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 thesis, we consider the case of burst deletion pattern, which tends to have bursts of consecutive 1's. Below is an example of how the output file string Y is generated from the input file string X and the burst deletion pattern D.

Example 1. *We have a file string* $X = (1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0)$*, and the burst deletion pattern* $D = (0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0)$*. Then we get the output file string* $Y(X, D) = (1, 0, 1, 0, 0, 0)$.

At last, we give out the problem of file synchronization from burst deletions as follows: Let node A contain a file that is represented by a binary string X of length n. Node B, which is far away from node A, contain a file string Y of length m. 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. That means that we do not know the contents and positions of the deleted bits. We suppose that the source file string X is generated by an i.i.d. Bernoulli source of parameter $\frac{1}{2}$, i.e., $P(X_i = 1) = 1 - P(X_i = 0) = 0.5$. And the burst deletion pattern is mathematically modeled as a stationary two-state Markov chain, which will be discussed in detail in Section 2.2. We are interested in a synchronization algorithm on a two-way, error-free channel between nodes A and B so that node B can recover file string X from Y with a small probability of error at the end of the communication.

Next, we will discuss how to use a stationary two-state Markov chain as a mathematical model for the desired burst deletion pattern.

2.2 The Burst Deletion Pattern

In practical applications, the deleted bits in the file string X usually appear in bursts. In order to model this 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 $i = 2, 3, ..., n$. Hence, the transition matrix of the Markov chain is denoted as $T = \begin{bmatrix} P(D_i = 0|D_{i-1} = 0) & P(D_i = 0|D_{i-1} = 1) \\ P(D_i = 1|D_{i-1} = 0) & P(D_i = 1|D_{i-1} = 1) \end{bmatrix} = \begin{bmatrix} p_1 & 1 - p_2 \\ 1 - p_1 & p_2 \end{bmatrix}$. 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 Figure 2.1. From the definitions of p_1 and p_2, we know that both of p_1 and p_2 should be very close to 1 to make the

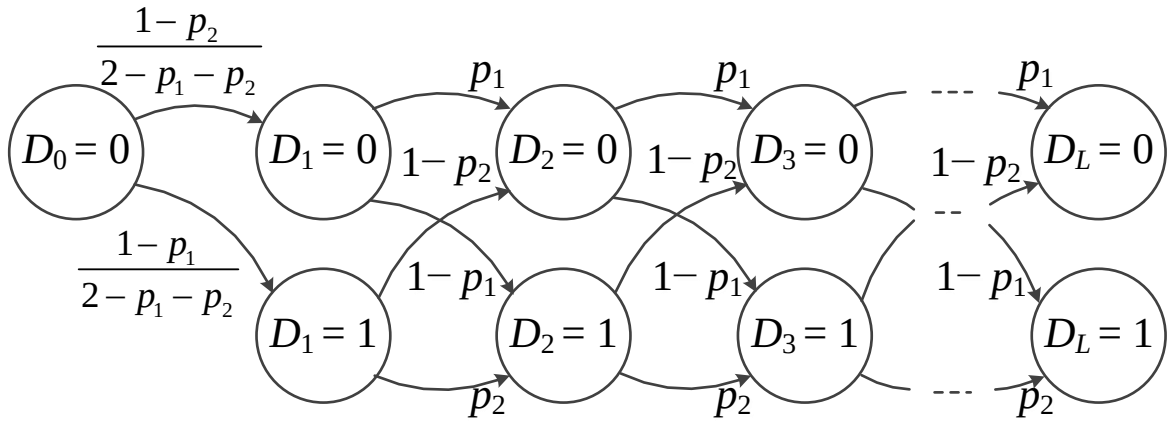

Figure 2.1: The proposed model for generating burst deletion pattern D. $D_i = 1$ means that the ith bit is deleted, and $D_i = 0$ means that it is not deleted.

deleted bits appear in bursts.

Next, we will show some properties of the proposed Markov chain model for the burst deletion pattern, which are useful for the mathematical analysis in later sections.

2.2.1 Properties

In some of previous works, the deletion pattern is the i.i.d. deletion pattern, which assumes that every bit of input file string X is deleted independently and with probability $\beta \ll 1$. From the property given below, we see that the i.i.d. deletion pattern can be also modeled by our stationary two-state Markov chain model under some appropriate parameter selections.

Property 1. *If $1 - p_1 = p_2 = \beta$, the deletion pattern modeled by the proposed stationary Markov chain model is i.i.d. deletion pattern with deletion probability β.*

Proof. According to the definition of Markov model,

$$\begin{aligned} P(D_i = 1) &= P(D_i = 1|D_{i-1} = 0)P(D_{i-1} = 0) + P(D_i = 1|D_{i-1} = 1)P(D_{i-1} = 1) \\ &= (1 - p_1)P(D_{i-1} = 0) + p_2 P(D_{i-1} = 1) \\ &= \beta(P(D_{i-1} = 0) + P(D_{i-1} = 1)) \\ &= \beta = P(D_i = 1|D_{i-1} = 0) = P(D_i = 1|D_{i-1} = 1) \end{aligned}$$

Similarly, we can get that

$$\begin{aligned} P(D_i = 0) &= P(D_i = 0|D_{i-1} = 0)P(D_{i-1} = 0) + P(D_i = 0|D_{i-1} = 1)P(D_{i-1} = 1) \\ &= p_1 P(D_{i-1} = 0) + (1 - p_2)P(D_{i-1} = 1) \\ &= (1 - \beta)(P(D_{i-1} = 0) + P(D_{i-1} = 1)) \\ &= 1 - \beta = P(D_i = 0|D_{i-1} = 0) = P(D_i = 0|D_{i-1} = 1) \end{aligned}$$

Hence, $P(D_i) = P(D_i|D_{i-1})$. D_i is independent with D_{i-1}, and they have the same distribution with $P(D_i = 1) = 1 - P(D_i = 0) = \beta$. □

Remark 1. *From Property 1, the proposed Markov chain model is suitable to generate both the i.i.d. deletion pattern and the burst deletion pattern. However, the selection for parameters p_1 and p_2 is different. For the i.i.d. deletion pattern, we have that $1 - p_1 = p_2 = \beta \ll 1$. In contrast, for the burst deletion pattern, the parameters p_1 and p_2 should satisfy that $1 - p_1 \ll 1$ and $1 - p_2 \ll 1$. The main difference is about p_2. In the i.i.d. deletion pattern, every bit is deleted independently with a small probability β. Hence, the deleted bits should be uniformly distributed in the whole file string. It is unlikely that two deleted bits are adjacent, which means that p_2 is very small. However, for the burst deletion pattern, many deleted bits are adjacent to each other. So the parameter p_2 should be very close to 1.*

The next property demonstrates why the proposed Markov model is a good choice for the burst deletion pattern. In statistics, the Ising model [22], which is a prototypical Markov random field, is often used to generate a class of signals with nonzero entries distributed in bursts. Since our Markov chain model is one-dimensional, we only consider one-dimensional Ising model here. For a signal $\mathbf{x} \in \mathcal{R}^n$, its support is $\mathbf{s} = \mathbf{sp}(\mathbf{x}) \in \{-1, 1\}^n$, in which $s_i = -1$ for $x_i = 0$ and $s_i = 1$ for $x_i \neq 0$. Then, if $\mathbf{s}$ is distributed based on a one-dimensional Ising model, its probablity density function (pdf) is as follows:

$$p(\mathbf{s}; \lambda, \lambda') = \mathbf{exp}\left\{\sum_{i=1}^{n-1} \lambda'_i s_i s_{i+1} + \sum_{i=1}^{n} \lambda_i s_i - Z_{\mathbf{s}}(\lambda, \lambda')\right\}, \tag{2.1}$$

in which $\lambda'_i > 0$ denotes the contribution from the relation of two adjacent entries s_i, s_{i+1}, λ_i is the contribution of every s_i, and $Z_{\mathbf{s}}(\lambda, \lambda')$ is a strictly convex function with respect to

λ and λ' that normalizes the distribution so that it integrates to one. In general, the pdf of the Ising model will be much large when $\mathbf{s}$ have bursts of consecutive -1's and 1's. Then, we have the following property:

Property 2. *The proposed Markov chain model for burst deletion pattern is a special case of the Ising model. Specifically, when the support* $\mathbf{s}$ *in the Ising model and the deletion pattern* D *in the proposed Markov chain model have the following mapping relation*

$$s_i = \begin{cases} 1, & D_i = 1 \\ -1, & D_i = 0 \end{cases}$$

and the parameters of the Ising model is set as

$$\lambda_i = \begin{cases} \frac{1}{4}\ln\frac{p_2(1-p_1)}{p_1(1-p_2)}, & i = 1, n; \\ \frac{1}{2}\ln\frac{p_2}{p_1}, & i = 2, ..., n-1, \end{cases}$$
$$\lambda'_i = \frac{1}{4}\ln\frac{p_1 p_2}{(1-p_1)(1-p_2)},\ i = 1, ..., n-1,$$

the Ising model degenerates to the proposed Markov chain model, which is equipped by concise and meaningful parameters.

Proof. First, we define $b_i = \exp(\lambda'_i)$ and $a_i = \exp(\lambda_i)$, and rewrite the pdf of the one-dimensional Ising model (2.1) as

$$p(\mathbf{s}; \mathbf{a}, \mathbf{b}) = C \prod_{i=1}^{n-1} b_i^{s_i s_{i+1}} \prod_{i=1}^{n} a_i^{s_i} \tag{2.2}$$

Then from the definition of the proposed Markov chain model, it is also known that

$$p(s_1) = \begin{cases} \frac{1-p_2}{2-p_1-p_2}, & s_1 = -1; \\ \frac{1-p_1}{2-p_1-p_2}, & s_1 = 1, \end{cases} \tag{2.3}$$

$$p(s_i = -1 | s_{i-1} = -1) = p_1,\ i = 2, 3, ..., n \tag{2.4}$$

$$p(s_i = 1 | s_{i-1} = 1) = p_2,\ i = 2, 3, ..., n \tag{2.5}$$

Our goal is to get the expressions of a_i's and b_i's and make (2.2) equal to (2.3), (2.4) and (2.5). We first consider a_i's and b_i's for $i = 2, 3, ..., n-1$.

for any three consecutive bits (s_{i-1}, s_i, s_{i+1}), we change its value from $(-1, 1, -1)$ to $(-1, -1, -1)$. Then (2.2) is $a_i^{-2} b_i^2 b_{i-1}^2$ times of its original value. From (2.3), (2.4) and (2.5), we get that

$$a_i^{-2} b_i^2 b_{i-1}^2 = \frac{p_1^2}{(1-p_1)(1-p_2)} \tag{2.6}$$

Similarly, we change (s_{i-1}, s_i, s_{i+1}) from $(1, -1, 1)$ to $(1, 1, 1)$, we have that

$$a_i^2 b_i^2 b_{i-1}^2 = \frac{p_2^2}{(1-p_1)(1-p_2)} \tag{2.7}$$

Comparing the above two equations (2.6) and (2.7), we can derive that $a_i^4 = p_2^2/p_1^2$, i.e., $a_i = \sqrt{p_2/p_1}$ (for $i = 2, 3, ..., n-1$). We further change (s_{i-1}, s_i, s_{i+1}) from $(-1, 1, 1)$ to $(-1, -1, 1)$, it is known that

$$a_i^{-2} b_i^{-2} b_{i-1}^2 = \frac{p_1}{p_2} \tag{2.8}$$

Comparing (2.6) and (2.8), it can be seen that $b_i^4 = p_1 p_2/(1-p_1)(1-p_2)$, i.e., $b_i = \sqrt[4]{p_1 p_2/(1-p_1)(1-p_2)}$ (for $i = 2, 3, ..., n-1$).

Following the similar approach, we can change the value of (s_1, s_2) and (s_{n-1}, s_n) and get the expressions of a_1, a_n and b_1. After getting all the a_i's and b_i's, it can be easily verified that (2.2) is always consistent with (2.3), (2.4) and (2.5). Hence, the Ising model is equivalent to the proposed Markov chain model in this case. □

Remark 2. *In general, the parameters of the Ising model λ, λ' are not unknown. Some literature utilizes some statistical methods to decide these two parameters. By our proposed Markov chain model, we use p_1 and p_2 to decide the unknown parameters λ, λ' in the Ising model, and hence can get the burst deletion pattern with high probability.*

Then, we need to investigate how the parameters p_1 and p_2 in the Markov chain model affect the overall deletion rate and the average length of bursts of non-deleted bits and deleted bits. The following property answers this question, which is as follows:

Property 3. *For the burst deletion pattern generated by the proposed Markov model, the average deletion rate, the average length of bursts of non-deleted bits and deleted bits, which*

are denoted by $\overline{d}$, $\overline{B_{nd}}$, *and* $\overline{B_d}$ *respectively, follow*

$$\overline{d} = \frac{1-p_1}{2-p_1-p_2}, \tag{2.9}$$

$$\overline{B_{nd}} = \frac{1}{1-p_1}, \tag{2.10}$$

$$\overline{B_d} = \frac{1}{1-p_2} \tag{2.11}$$

Proof. Since the proposed Markov model is already in the steady state, the probability of $D_i = 1$ for $i = 1, 2, ..., n$ can be gotten as $(1-p_1)/(2-p_1-p_2)$. That means that every bit has the probability of $(1-p_1)/(2-p_1-p_2)$ to be deleted on average. Then, the average deletion rate for the whole file string is still $(1-p_1)/(2-p_1-p_2)$.

Next, we show how the average length of bursts of non-deleted bits and deleted bits $\overline{B_{nd}}$ and $\overline{B_d}$ are derived. For the probability of the length of a burst of deleted bits being l, which is denoted as $P(B_d = l)$, we can have the following equations:

$$P(B_d = l) = P(B_d = l-1) * p_2$$

Then we also know that the sum of $P(B_d = l)$ for all $l = 1, 2, ...$ is 1, i.e.,

$$\sum_{l=1}^{+\infty} P(B_d = l) = 1$$

From the above two equations, we know that the length of a burst of deleted bits follows a geometric distribution $P(B_d = l) = (1-p_2)p_2^{l-1}$. Hence, the average length of bursts of deleted bits $\overline{B_d}$ is $1/(1-p_2)$.

Similarly, we can know that the probability of the length of a burst of non-deleted bits being l, which is denoted as $P(B_{nd} = l)$, satisfies the following two equations:

$$P(B_{nd} = l) = P(B_{nd} = l-1) * p_1$$

$$\sum_{l=1}^{+\infty} P(B_{nd} = l) = 1$$

We can also get that the length of a burst of non-deleted bits still follows a geometric distribution $P(B_{nd} = l) = (1-p_1)p_1^{l-1}$. And the average length of bursts of non-deleted bits $\overline{B_{nd}}$ is $1/(1-p_1)$. □

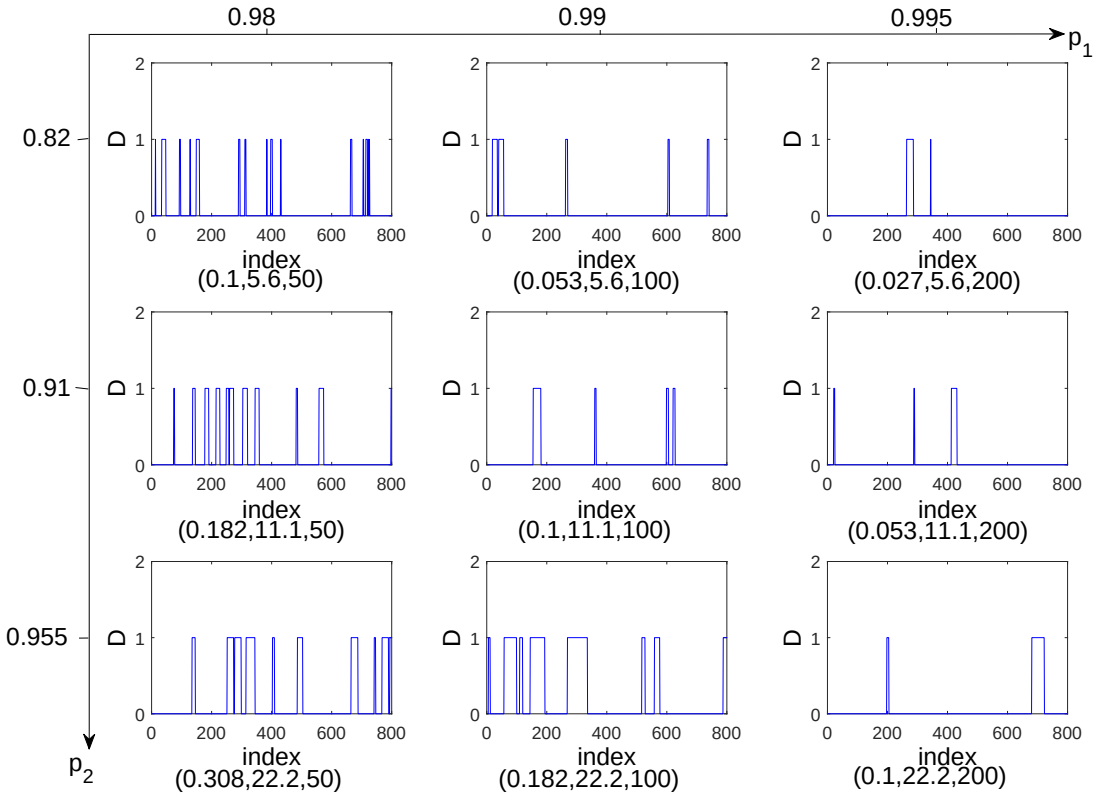

Figure 2.2: The examples of burst deletion patterns generated by the proposed Markov model with $n = 800$ and various (p_1, p_2). $(\overline{d}, \overline{B_d}, \overline{B_{nd}})$ with corresponding parameter set is listed below each subfigure. The three burst deletion patterns located on the diagonal subplots share the same average deletion rate.

Several examples of burst deletion pattern generated by $n = 800$, and various (p_1, p_2) are shown in Figure 2.2. Inside every row and every column, the average burst length of deleted and non-deleted bits increase, respectively, with respect to p_2 and p_1. Moreover, the three burst deletion patterns located on the diagonal subplots satisfy $\overline{d} = 0.1$ and have 80 expected deleted bits.

At last, we will present a property of the Markov chain model, which tells us how to get the probability of δ deleted bits within L consecutive bits.

Property 4. *For a burst deletion pattern generated by the proposed Markov model, within*

any L consecutive bits, the probability of the number of deleted bits $\Delta_L = \delta$ is

$$P(\Delta_L = \delta) = \begin{cases} \dfrac{1-p_2}{2-p_1-p_2}p_1^{L-1} & \delta = 0 \\ \dfrac{1}{(2-p_1-p_2)(1-p_1)}\Big(Q_{L-\delta,\delta} + 2(1-p_1-p_2)Q_{L-\delta-1,\delta} & \\ \quad + (1-p_1-p_2)^2 Q_{L-\delta-2,\delta}\Big) & 1 \le \delta \le L \end{cases} \tag{2.12}$$

where

$$Q_{j,m} = \begin{cases} (1-p_1)^{m+1}\displaystyle\sum_{n=0}^{m-1}\binom{m-1}{n}\binom{m+j-n}{m-n}(\frac{p_1+p_2-1}{1-p_1})^n(1-p_2)^{m-1-n}p_1^{j-m+1} & j \ge 0 \\ 0 & j < 0 \end{cases} \tag{2.13}$$

When $1-p_1 = p_2 = \beta$, the deletion pattern generated by the proposed Markov model becomes i.i.d. with deletion rate β (This can be seen from Property 1). And (2.12) *is equivalent to*

$$P(\Delta_L = \delta) = \binom{L}{\delta}\beta^\delta(1-\beta)^{L-\delta}, \tag{2.14}$$

which is the probability of δ deleted bits with L consecutive bits in the i.i.d. deletion pattern.

Proof. We denote the deletion pattern for any consecutive L bits as $(D_i, D_{i+1}, ..., D_{i+L-1})$. When the number of deleted bits Δ_L is 0, we can easily get that

$$\begin{aligned} P(\Delta_L = 0) &= P(D_i = 0)\ P(D_{i+1} = 0|D_i = 0)\ ...\ P(D_{i+L-1} = 0|D_{i+L-2} = 0) \\ &= \frac{1-p_2}{2-p_1-p_2}p_1...p_1 \\ &= \frac{1-p_2}{2-p_1-p_2}p_1^{L-1} \end{aligned}$$

However, for the case that $\Delta_L > 0$, the derivation becomes very complicated. We omit the complete proof here. For more details, please refer to [21]. When $1-p_1 = p_2 = \beta$, we just insert p_1 and p_2 into (2.12) and the result of (2.14) can be easily derived. □

Above all, we find that the proposed Markov model for the burst deletion pattern has many useful properties. From property 2, it shows that the Markov model is suitable to generate the desired burst deletion pattern. There are still topics need to be studied on the

stationary two-state model. For example, it is important to build the relation between the model and the real-world deletion patterns, and to estimate the parameters p_1 and p_2 from specific scenario. However, this is beyond the scope of this thesis and is not included.

2.2.2 Assumptions

In order to simplify the file synchronization problem, we make two assumptions for the burst deletion pattern:

(1) The length of every burst of deleted bits should be small (e.g., any length between 2 and 10 should be fine).

(2) The average percentage of the deleted bits is small (e.g., it might be 0.005, 0.01).

For assumption (1), it should be satisfied that $1/(1-p_2)$ is a small positive number (e.g., a number smaller than 10). This can be seen from Property 3. In Property 3, we know that the average length of bursts of deleted bits $\overline{B_d} = 1/(1-p_2)$. And the length of a burst of deleted bits follows a geometric distribution. Hence, the length of most bursts of deleted bits should be around $1/(1-p_2)$. There are more bursts with length smaller than $1/(1-p_2)$ than bursts with length larger than $1/(1-p_2)$. As for assumption (2), still from Property 3, we get that the average percentage of the deleted bits is $(1-p_1)/(2-p_1-p_2)$. This means that $1-p_1 \ll 1-p_2$.

Hence, due to the above two assumptions, we have the following requirements for the parameters p_1 and p_2 of the proposed Markov model.

(1) p_2 should satisfy that $1/(1-p_2)$ is a positive number smaller than 10.

(2) For every fixed p_2 (i.e., the average length of deletion bursts is constant), we consider the case of various p_1, which satisfies that $1-p_1 \ll 1-p_2$.

At the last of this subsection, we gave out two examples of the burst deletion pattern, which are generated by the proposed Markov model and satisfy the above two requirements for the parameters. We first set (p_1, p_2) as $(0.9982, 0.82)$. In this case, we have that $1/(1-$

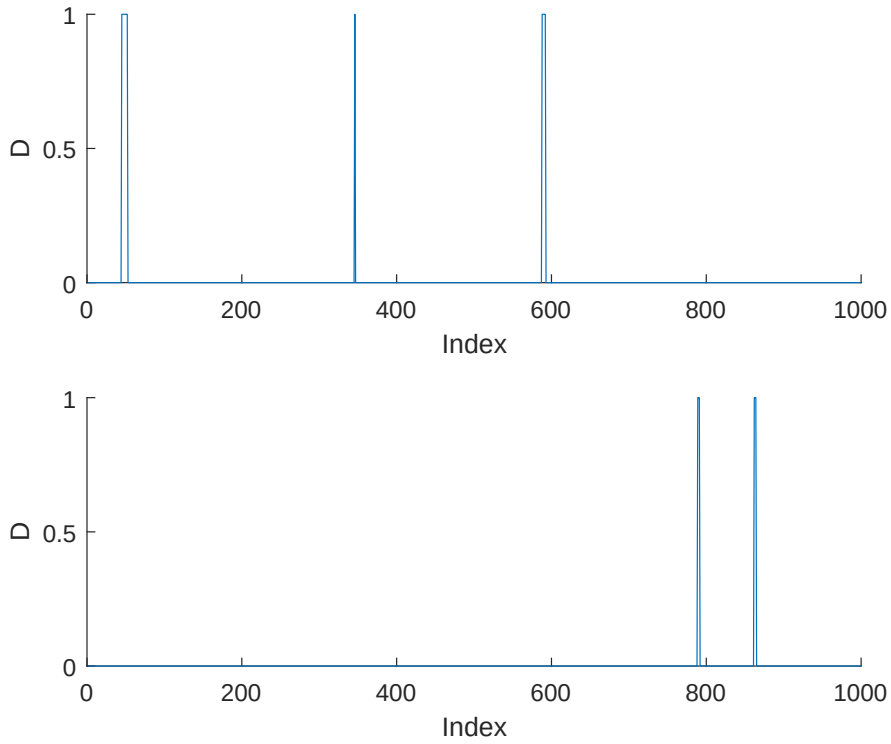

Figure 2.3: Two examples of the burst deletion pattern generated by the proposed Markov model. The parameter settings of (p_1, p_2) for the above one and the below one are $(0.9982, 0.82)$ and $(0.9991, 0.82)$, respectively.

$p_2) = 5.56 < 10$ and $1 - p_1 \ll 1 - p_2$. The average deletion rate is 0.01. One example of the deletion pattern in this setting is shown in the above subfigure of Figure 2.3. It can be seen that there are 3 deletion bursts, which are located at $[45, 46, 47, 48, 49, 50, 51, 52]$, $[346]$ and $[588, 589, 590, 591, 592]$. All of the deletion bursts are of small length. And only a small percentage of bits are deleted. Then, we change the setting of (p_1, p_2) to $(0.9991, 0.82)$, which still satisfies the two requirements for p_1 and p_2. At this time, the average deletion rate becomes 0.005. The example is shown in the below subfigure of Figure 2.3. There are 2 deletion bursts located at $[789, 790, 791]$ and $[862, 863, 864]$.

CHAPTER 3

Existing Protocol Review

Since the synchronization protocol in [14] is very inspiring for the construction of the new protocol, we present a review of the synchronization protocol in [14] before the new synchronization algorithm is formally introduced. This protocol is consist of three modules: the matching module, the deletion recovery module, and the LDPC decoder module. We first take an overview of these three modules to know their individual roles in the whole synchronization protocol. Also, some theoretical results are given out, which guarantee that the protocol has order-optimal rate and its bit error probability is exponentially small in the size of X. Then, by inspecting each module in more details, we will have a deeper understanding of their implementation.

The problem setting of the synchronization protocol in [14] is same with ours, except for the only difference in deletion pattern. In [14], the deletion pattern is i.i.d., which means that every bit of the source file X is deleted independently and with the same probability $\beta \ll 1$. Hence, the deleted bits are expected to be distributed uniformly in the whole file string. The purpose of the synchronization protocol is to efficiently divide the files X and Y into matching correlated segments so that each of matching segments only contain one deletion. Then each matching segment can be synchronized by VT codes.

3.1 Protocol Overview

In this section, we provide an overview of the synchronization protocol. A graphical illustration of this protocol is presented in Figure 3.1. We can see that there are three modules within the protocol: the matching module, the deletion recovery module, and the LDPC

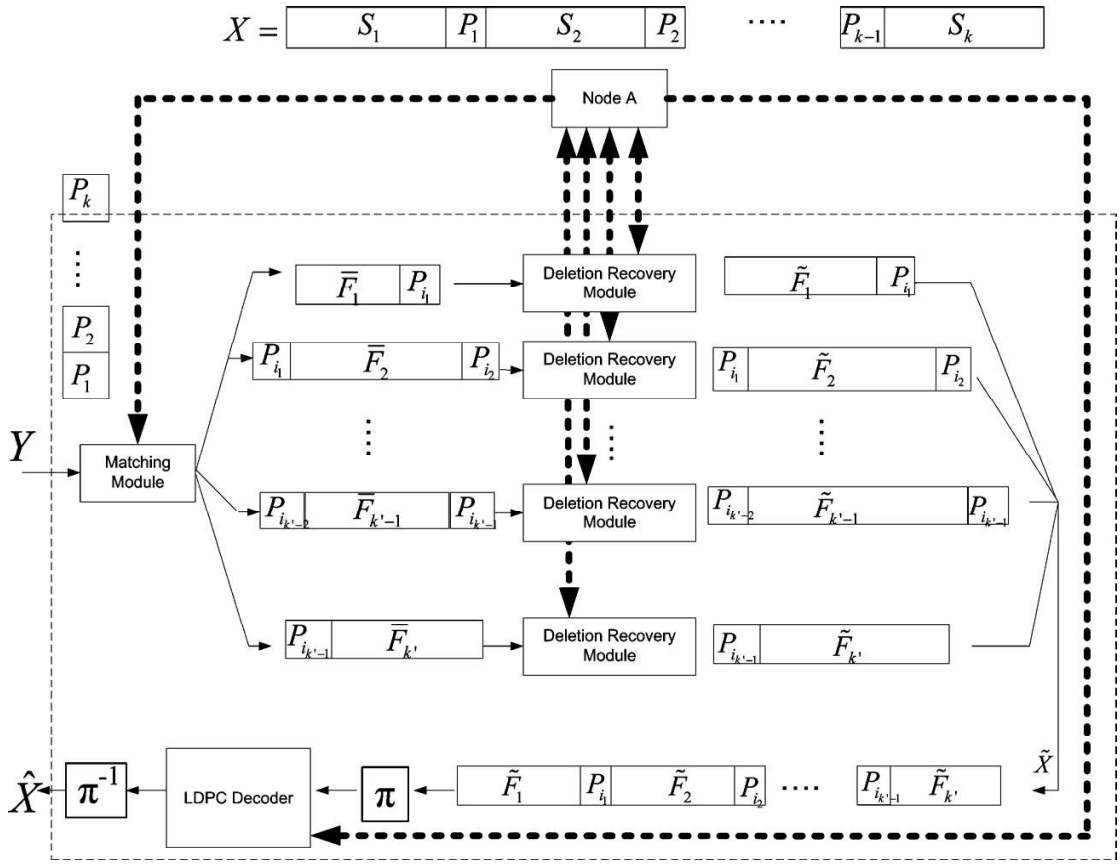

Figure 3.1: (Yazdi, [14]) Illustration of the synchronization protocol. The original string X is broken up into segment substrings S_i, and pivot substrings P_i. User A sends the pivot strings to the matching module, which matches them in the deleted string Y as P_{i_j}. Between the matched pivots are the segments $\overline{F}_i$. It is the goal of the deletion recovery module to synchronize these strings to the S_i. The results are sent to the LDPC decoder module, which corrects errors introduced in the first two modules and produces the final reconstructed $\hat{X}$.

decoder module. These three module work in series, such that the input to the first module is string Y and the output of the last module is the estimate $\hat{X}$ of string X. There is no interactivity between the two nodes for the matching module and the LDPC decoder module. The interactivity only occurs in the deletion recovery module.

To begin, node A divides the file X into *segment* substring $S_i (1 \leq i \leq k)$ and *pivot*

substring $P_i (1 \leq i \leq k-1)$. Hence, the source file X is partitioned as follows:

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

where $|P_i| = L_P$ and $|S_i| = L_S$. We set $L_S = 1/\beta$ and $L_P = O(\log(1/\beta))$, and both nodes A and B know the exact values of L_S and L_P. Note that the length of a pivot string L_P is much smaller than the length of a segment string L_S. The length of the segment string L_S is selected in such a way that the expected number of deletions within each segment string is on the order of 1. For the pivot string, its length is selected to be short enough that with high probability they do not contain any deletions, but long enough to ensure that there are very few copies of the pivots in the whole file string.

3.1.1 Matching Module

The first step of the synchronization protocol is performed by the matching module at node B. As we can see from Figure 3.1, node A sends pivot strings P_1, P_2,..., P_{k-1} to node B in sequential order. After receiving all the pivot strings, the matching module attempts to find the exact copies of P_i's within file Y. There are three possible outcomes for each of the P_i's:

(1) The matching module is able to successfully find the match corresponding to P_i in file Y;

(2) Any match for P_i is not found by the maching module;

(3) A match which does not correspond to the transmitted P_i is erroneously found.

We see that there are two cases for the unsuccessful match of P_i. The case of no matching of P_i is due to the possible deletions within the pivot string. For the other case, it is because that there are multiple matches for a pivot and it is not very apparent which match is the correctly matched one. Hence, the matching module is responsible for resolving ambiguities of multiple matches, and uses a graph-theoretic approach to find the most likely matches of the pivot strings in Y based on the fact that the pivot strings were sent in order from file X.

Since the matching module is only able to find the matches for a subsets of P_i's, we denote these P_i's 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 [1], which accordingly divides file X into

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

where F_j is the substring between pivot $P_{i_{j-1}}$ and P_{i_j} in X and can be written as

$$F_j = S_{i_{j-1}+1}, P_{i_{j-1}+1}, ..., P_{i_j-1}, S_{i_j}.$$

Notice that if $P_{i_{j-1}}$ and P_{i_j} are matched correctly in Y, $\overline{F}_j$ can be derived from F_j by some deletions. The detailed implementation of the matching module will be discussed in Section 3.2.1.

3.1.2 Deletion Recovery Module

After the matching of pivots between file X and Y, the problem of synchronizing the long string Y with X is divided into multiple simpler problems of synchronizing $\overline{F}_j$ with the corresponding F_j. The goal of the deletion recovery module is to correct all the deletions in the segment string F_j. In order to finish this, it uses the synchronization protocol of Venkataramanan [12].

The key element of this synchronization protocol is the use of VT codes, which allows us to correct a single insertion or deletion in a binary string. The algorithm uses a divide-and-conquer approach. Every time we get an unsynchronized substring in Y, we compare its length with the corresponding substring in X. if they have the same length, we assume that they are synchronized. If their lengths differ by one, we just use the VT code to finish the synchronization. If the difference of their lengths is larger than one, the synchronization

[1] In practice, in order to make the interactive communication only occur at the deletion recovery module, we put off the sending of indices to the next module.

by using VT code is not possible. In this case, we divide this substring into two smaller substrings. This divide-and-conquer approach can iteratively reduce the size of the matched substrings until they can be synchronized by the VT code.

At the end of this step, the estimates of all F_j's are formed based on the corresponding $\overline{F}_j$'s. We denote these estimates of F_j's as $\tilde{F}_j$'s. Note that the length of $\tilde{F}_j$ is the same as that of F_j. Then, an estimate of the source file X is gotten as:

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

This $\tilde{X}$ is the final output of the deletion recovery module, and will be sent to the next module. We can see that $\tilde{X}$ has the same length as the source file X. The algorithm for the implementation of this module will be explained with more details in Section 3.2.2.

3.1.3 LDPC Decoder Module

At the last step, the LDPC decoder module at node B, corrects any errors made in the first two modules. There are two types of such errors. First, it is possible that the matching module matches a pivot P_i at a wrong position due to multiple copies of this pivot in Y. If this case occurs, substring $\overline{F}_j$ and $\overline{F}_{j+1}$ may differ from the corresponding F_j and F_{j+1} by a very large number of deletions. Then, the synchronization protocol of Venkataramanan will not work well and have a large error rate. Consequently, $\tilde{F}_j$ and $\tilde{F}_{j+1}$ may be very different from F_j and F_{j+1}, respectively. In addition, even two neighboring pivots are properly matched, the synchronization algorithm for the deletion recovery module is not error-free. It is possible for the algorithm to conclude that two substrings are synchronized when in fact they are not.

Suppose that the total error of the first two synchronization modules is bounded by ζ, i.e.,

$$P(\tilde{F}_j \neq F_j) \leq \zeta.$$

Since the error rate over substring $\tilde{F}_j$ is an upper bound for the bit error rate of file X, we can get that

$$P(\tilde{X}(i) \neq X(i)) \leq \zeta. \tag{3.1}$$

In order to recover from errors of $\tilde{X}$, we use an LDPC decoder, which receives parity check bits of a systematic LDPC code from node A. To avoid a potential nonuniformity of errors over different bits of $\tilde{X}$, we can apply a random permutation π and its inverse permutation π^{-1} at the input and the output of the LDPC decoder, respectively. Then, if the bit error rate is bounded as (3.1) and a sufficient number of parity check bits are sent to the LDPC decoder module, the output of the decoder will get a string $\hat{X}$ with

$$P(\hat{X}(i) \neq X(i)) \leq 2^{-\Omega(n)},$$

which means that the bit error rate is exponentially small in the size of file X.

So far, we have taken an overview of the synchronization protocol in [14]. Next, we will present some theoretical results, which demonstrates that the protocol has order-optimal rate and very small bit error rate.

3.1.4 Theoretical Results

In [14], the author gave out the following theorem:

Theorem 1 (Yazdi, [14])**.** *There exists a deterministic synchronization protocol between nodes A and B on a two-way, error-free channel, that on average transmits $O(n\beta \log \frac{1}{\beta})$ bits and generates an estimate $\hat{X} = \hat{X}(1), ..., \hat{X}(n)$ of X at node B, such that $P(\hat{X}(i) \neq X(i)) \leq 2^{-\Omega(n)}$ for every $1 \leq i \leq n$.*

The main contribution of [14] is to prove the above theorem. The result is very remarkable, since it proves that the protocol is optimal within a constant multiplicative factor for the case of i.i.d. deletion pattern. From the perspective of information theory, it is known that the optimal number of bits needed for the general problem of reconstructing a string X given a string Y is the conditional entropy of X given Y: $H(X|Y)$. By applying the result of [18] to i.i.d. deletion pattern, for a small value of β, the entropy $H(X|Y)$ can be estimated as

$$H(X|Y) = n(\beta \log \frac{1}{\beta} + O(\beta)).$$

Therefore, any optimal synchronization protocol would need at least $n(\beta \log \frac{1}{\beta} + O(\beta))$ transmitted bits, thus showing the optimality of our protocol. However, to achive the rate specified in the theorem, we need to know the parameter β ahead of time, which is not practical in real world applications.

In [15], the authors generalized the scenario in [14] in three way. First, both insertions and deletions are allowed for the edit channel (In [14], only deletion are allowed). Second, the file X is no longer binary. Every symbol within file X comes from a general alphabet $|\chi| = Q$. Third, the symbols within the file X is not drawn from χ uniformly. They can be drawn based on any generic distribution $\mu(x)$. Although this thesis also considers the case of deletion-only edits, the theoretical results of [15] might be still inspiring for the extension of our work. We give out the main theoretical result of [15] as follows:

Theorem 2 (Sala, [15])**.** *In the problem setting involving files selected according to i.i.d. (not necessarily uniform) distributions over arbitrary alphabets with fixed collision entropy $H_2 > 0$ affected by insertions and deletions, there exists a deterministic synchronization protocol between users A and B on a two-way, error-free channel, that on average transmits $O(\frac{nq}{H_2}\beta \log \frac{1}{\beta})$ bits and generates an estimate $\hat{X} = \hat{X}(1), ..., \hat{X}(n)$ of X at user B, such that $P(\hat{X}(i) \neq X(i)) \leq 2^{-\Omega(n)}$ for every $1 \leq i \leq n$.*

In the above theorem, $H_2 = -\log \sum_x \mu^2(x)$ is the collision entropy of string X, which denotes the probability of two independent samples of $\mu(x)$ being equal. q is defined as $q = \lceil \log Q \rceil$. And β represents the total rate of edits, which is the sum of the rate of insertions and deletions. Same with [14], to achieve this optimal communication rate requires the knowledge of β in advance.

3.2 Protocol Implementation

In this section, we will take a closer look at the implementation details of the matching module and the deletion recovery module.

3.2.1 Matching Module Implementation

The task of the matching module is to detect correct matches of P_i's within the file Y. In [14], a graph theoretic method is used to construct this module. The method make use of a matching graph to get the matches of pivots. Before we formally introduce this matching graph, we need to define the concepts of correct and incorrect matches.

3.2.1.1 Correct and Incorrect Matches

For each pivot string P_i in the file X, the deletion patterns that acts on them might be different. Hence, we consider the following three cases:

(1) There is no deletion within P_i. In this case, the corresponding copy of P_i within file Y is the correct match of P_i. All other copies of P_i in Y are considered incorrect matches of P_i.

(2) There is at least two deletions within P_i. For this case, all copies of P_i within file Y are considered incorrect matches.

(3) There is exactly one deletion within P_i. If this deletion can be seen as a deletion in the neighboring segment string and no deletion in P_i (i.e., the deletion can be "moved" to the neighboring segment string), we call the corresponding copy of P_i in Y a correct match of P_i and all other copies of P_i are incorrect matches of P_i. If this deletion can not be "moved" to the neighboring segment string, all copies of P_i within Y are called incorrect matches.

For the cases of no deletion and at least two deletions, the definition of correct and incorrect matches is very natural. Next, we will explain the definition of correct and incorrect matches for the case of exact one deletion in more detail. Let us take a look at the following three examples.

Example 2. *in X, we have* $..., 1, \underbrace{\lfloor 0, 1, 1, 0, 1, 0, \times 0, 0, \rfloor}_{P_i} 0, ...,$ *in which the penultimate bit is*

deleted from P_i. Then, in Y, we get the corresponding substring as $..., 1, \underbrace{\big|0, 1, 1, 0, 1, 0, 0, \big|}_{P_i} 0, ...$

We can see that the deletion within P_i can (only) be moved to the segment string that is after P_i. Hence, there is one correct match for P_i. All the other copies of this P_i in Y are considered as incorrect matches.

Example 3. *in X, we have* $..., 0, \underbrace{\big|0, 0, 0, 0, 0, 0, \times 0, 0, \big|}_{P_i} 0, ...,$ *in which the penultimate bit is deleted from P_i. Then, in Y, we get the corresponding substring as* $..., 0, \underbrace{\big|0, 0, 0, 0, 0, 0, 0, \big|}_{P_i} 0, ...$ *(or* $..., 0, \underbrace{\big|0, 0, 0, 0, 0, 0, 0, \big|}_{P_i} 0, ...$*).*

We can see that the deletion within P_i can be moved to the segment string that is after P_i or the segment string that is before P_i. In this case, there are two correct matches for P_i. All the other copies of this P_i in Y are considered as incorrect matches.

Example 4. *in X, we have* $..., 1, \underbrace{\big|0, 1, 1, 0, 1, 0, \times 0, 0, \big|}_{P_i} 1, ...,$ *in which the penultimate bit is deleted from P_i. Then, in Y, we get the corresponding substring as* $..., 1, \big|0, 1, 1, 0, 1, 0, 0, \big|1, ...$

It can seen that the deletion within P_i can not be moved to the neighboring segment string. Hence, there is no correct match for P_i.

Notice that there might be zero, one or two correct matches for P_i in the case of exact one deletion within P_i. In the case of at least two deletions within P_i, it might still be possible to move the deletions from P_i to the neighboring segment string. However, the probability of these cases is very small and hence we count these matches as incorrect matches.

At last, we present some theoretical results about the occurrence of correct matches for P_i's. Defining $R = 1 - L_P\beta + 2\beta$, we have the following three theorems.

Theorem 3 (Yazdi, [14])**.** *For a random string X and a random deletion pattern D, on average, the number of pivots with at least one correct match in Y is $(R + o(\beta))k$.*

Theorem 4 (Yazdi, [14])**.** *For a random string X and a random deletion pattern D, with probability $1 - 2^{-\Omega(n)}$, there are $(R + o(\beta))k$ pivots with at least one correct match in Y.*

Theorem 5 (Yazdi, [14]). *For a random string X and a random deletion pattern D, with probability $1 - 2^{-\Omega(n)}$, there are $o(\beta)k$ pivots with two correct matches in Y.*

From the above theorems, we can get that most of P_i's has at least one correct match. This inspire us to construct an efficient matching module, which can detect the correct matches of these P_i's.

3.2.1.2 Matching Graph

Now, we begin to construct the matching graph. We define a graph $G(V, E)$ which has $k+1$ layers of vertices denoted as $\Lambda_0, \Lambda_1, ..., \Lambda_k$. Each vertex in layer Λ_i, $1 \leq i \leq k-1$, represents a match of pivot P_i in Y. We further introduce two auxiliary vertices s and t, where $\Lambda_0 = \{s\}$ and $\Lambda_k = \{t\}$. Vertices s and t represent the start and the end of string Y, respectively.

We call a vertex in Λ_i a good (bad) vertex if it corresponds to a correct (incorrect) match of P_i within Y. In order to detect the correct matches of P_i's, we need to find good vertices in graph G. For that, we define an edge set E such that the good vertices are distinguished by their connectivity in the graph.

For two vertices $u \in \Lambda_i$ and $v \in \Lambda_j$ with $i < j$, we define the distance between u and v, which is denoted as $Dis(u, v)$, as the number of bits between the matches of two pivots that correspond to u and v. Since the edits are limited to deletions, the distance between any two good vertices u and v are at most $(j - i - 1)L_P + (j - i)L_S$. Also, it can be easily seen that the least value of the distance between two good vertices u and v is -1.

Hence, we get the edge set E of graph G as follows. For any two vertices $u \in \Lambda_i$ and $v \in \Lambda_j$ with $i < j$, we connect these two vertices if and only if

$$-1 \leq Dis(u, v) \leq (j - i - 1)L_P + (j - i)L_S. \tag{3.2}$$

Therefore, all pairs of good vertices from different layers are connected together. By definition, we know that s and t are auxiliary good vertices. Hence, good vertices across different layers form an s-t path in graph G. Nevertheless, there are potentially many other pairs of vertices that satisfy the condition (3.2) and are connected together. An example of graph G

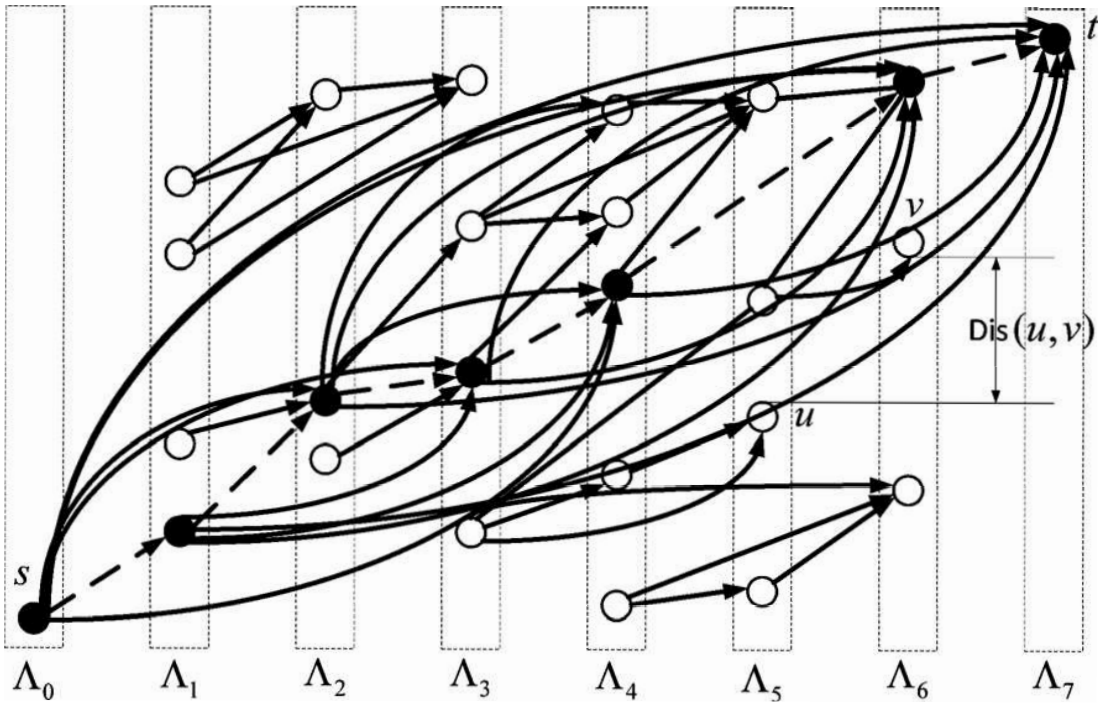

Figure 3.2: (Yazdi, [14]) Graph G with eight layers of vertices. The horizontal axis indicates different layers and the vertical axis indicates the position of each vertex in string Y that can take values from 1 to $|Y|$. The good and bad vertices are distinguished by black and white colors, respectively. The first layer has only one vertex s and the last layer has only one vertex t. As it is seen, all good vertices in the graph are connected together and they form an $s-t$ path, which is represented by the dashed edges in the graph.

with eight layers and its edge set defined by (3.2) is shown in Figure 3.2.

In [14], the authors prove that any $s-t$ path of appropriate length in grapg G is formed mostly of good vertices with a very high probability, which is stated as follows:

Theorem 6 (Yazdi, [14])**.** *Let X be a random input string to a deletion channel and D be a random deletion pattern. Let Y be the string obtained from X and D. Let G denote the matching graph corresponding to Y. Then, for $L_P \geq 11 + 2\log\frac{1}{\beta}$, with probability at least $1 - 2^{-\Omega(n)}$, all paths from s to t with $(1 - L_P\beta + 2\beta)k + o(\beta)k$ vertices, have at least*

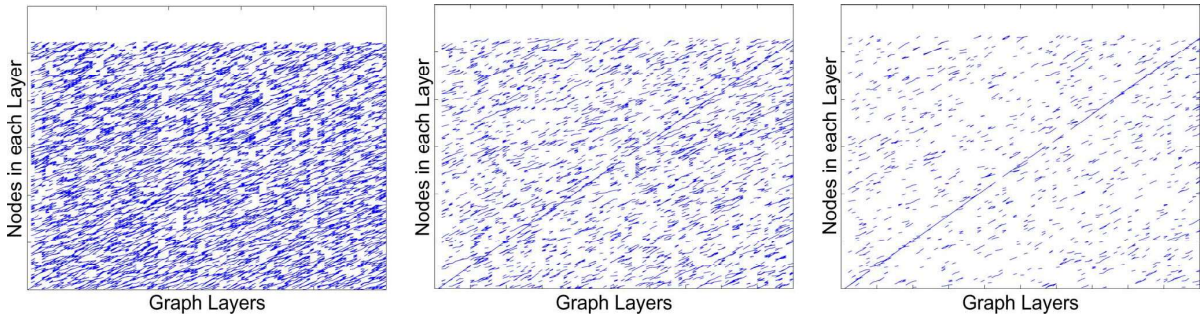

Figure 3.3: (Yazdi, [14]) A matching graph in the practical setting. The parameters are set as $k = 100$, $\beta = 0.01$, $L_S = 100$, and $L_P = 6, 7, 8$. Only the edges between consecutive layers are depicted.

$(1 - L_P\beta + 2\beta)k + o(\beta)k - \beta k$ *good vertices.*

The above theorem has a very meaningful algorithmic implication. If we find any $s - t$ path with $(1 - L_P\beta + 2\beta)k + o(\beta)k$ vertices, we get many good vertices on this path. Since finding such a path is computationally tractable, getting a large fraction of good vertices (i.e., correct matches of pivots) is also a tractable task. A matching graph in the practical setting is presented in Figure 3.3, which verifies the result of the above theorem. As it is shown from the figure, for small L_P, there are many edges in the graph and potentially many $s - t$ paths do not share many vertices with the correct path. When L_P is larger, many irrelevant edges disappear and the only remaining path is the one formed by most good vertices. We observe that $L_P = 8$ is sufficient to detect good vertices on the matching graph G.

3.2.1.3 Practical Implementation

Next, we discuss the explicit implementation of the graph-based algorithm for the matching module. From Theorem 6, we can see that it is enough to find an $s-t$ path with $(1-L_P\beta+2\beta)k+o(\beta)k$ vertices (with a not small L_P) to detect a large number of correct matches for pivots in file Y. This problem can be considered as a shortest path problem in a directed graph, which is solved in polynomial time. The detailed procedure is as follows.

First, we keep only the vertices that have an edge to vertex t and remove all other vertices. This step does not eliminate any good vertex since all good vertices are connected to vertex t. We denote the resulting graph as $\tilde{G}$. Then, we try to find the longest $s-t$ path in $\tilde{G}$. Because all good vertices are connected together and form an $s-t$ path of length $(1-L_P\beta+2\beta)k+o(\beta)k$, the longest path in $\tilde{G}$ has at least $(1-L_P\beta+2\beta)k+o(\beta)k$ vertices. At last, we keep only the first $(1-L_P\beta+2\beta)k+o(\beta)k$ vertices in the longest path. Since each vertex in $\tilde{G}$ is connected to vertex t, the resulting vertices from this step form a path with $(1-L_P\beta+2\beta)k+o(\beta)k$ vertices from s to t. As shown in [14], the computational complexity of the matching map is upper bounded by $O(n^4\beta^6)$.

3.2.2 Deletion Recovery Module Implementation

Once the pivots have been matched, the problem of synchronizing two large files X and Y is divided into independent synchronizations of each short segment string. The goal of the deletion recovery module is to correct all the deletions within each segment string. It uses the algorithm in [12], which is suitable for our case. Since a key component of the algorithm in [12] is a one-way single deletion correcting algorithm, we first give out an introduction of it before presenting the algorithm of [12].

3.2.2.1 Synchronizing From One Deletion

Here, we describe how to optimally synchronize from a single deletion. This one-way single deletion synchronization algorithm [2] is based on the use of VT codes, which is defined as follows:

Definition 1. *For code length n and integer $a \in \{0,1,...,n\}$, the VT code $VT_a(n)$ consists of all binary vectors $X=(x_1,x_2,...,x_n)$ satisfying*

$$\sum_{i=1}^{n} ix_i \equiv a \bmod (n+1)$$

[2]Although this algorithm is also able to correct a single insertion with a small modification, we ignore this part since we only focus on the deletion-only case.

An example of VT code with $n = 5$ and $a = 0$ is:

Example 5. *$VT_0(5)$ is consist of the following codes:*

$$VT_0(5) = \{(x_1, x_2, x_3, x_4, x_5) : \sum_{i=1}^{5} ix_i \bmod 6 = 0\}$$
$$= \{00000, 10001, 01010, 11011, 11100, 00111\}.$$

For any $a \in \{0, 1, ..., n\}$, the code $VT_a(n)$ can be used to communicate reliably over a channel that introduces one deletion. The decoding algorithm for $VT_a(n)$, which is proposed by Levenshtein in [3], is reproduced below.

1. Suppose that a codeword $X \in VT_a(n)$ is transmitted, the channel deletes the bit in position p, and 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 $p = 1 + L_0 + L_1$).

2. The channel decoder computes the weight of Y given by $wt(Y) = L_1 + R_1$, and the new checksum $\sum_i iy_i$. If the deleted bit is 0, the new checksum is smaller than the checksum of X by an amount R_1. If the deleted bit is 1, the new checksum is smaller by an amount $p + R_1 = 1 + L_0 + L_1 + R_1 = 1 + wt(Y) + L_0$.
   Define the *deficiency* $D(Y)$ of the new checksum as the amount by which it is smaller than the next larger integer of the form $k(n + 1) + a$, for some integer k. Thus, if a 0 was deleted the deficiency $D(Y) = R_1$, which is less than $wt(Y)$; if a 1 was deleted $D(Y) = 1 + wt(Y) + L_0$, which is greater than $wt(Y)$.

3. If the deficiency $D(Y)$ is less than or equal to $wt(Y)$, the decoder determines that a 0 was deleted, and restores it just to the left of the rightmost R_1 1's. Otherwise a 1 was deleted and the decoder restores it just to the right of the leftmost L_0 0's.

To illustrate the above decoding procedure, we give out two examples as follows.

Example 6. *Assuming that $X = (1, 0, 0, 0, 1) \in VT_0(5)$ is transmitted over the channel. And the second bit in X is deleted, which results in $Y = (1, 0, 0, 1)$. The checksum of Y is 5, and the deficiency $D = 6 - 5 = 1 < wt(Y) = 2$. The decoder inserts a 0 to the left of rightmost $D = 1$ 1's and get $(1, 0, 0, 0, 1)$.*

Example 7. *We still assume that the transimitted code is* $X = (1, 0, 0, 0, 1) \in VT_0(5)$. *We delete the last bit of* X *and get* $Y = (1, 0, 0, 0)$. *Then the new check sum of* Y *is* 1, *and the deficiency* $D = 6 - 1 = 5 > wt(Y) = 1$. *The decoder inserts a* 1 *to the right of the leftmost* $D - 1 - wt(Y) = 3$ 0*'s and get* $(1, 0, 0, 0, 1)$.

It should be noted that the 0 is restored in the fourth position while the original deleted bit is in the second position. The decoding algorithm of VT code exploits the fact that a deleted bit can be restored at any position within the correct run. And it always restores a deleted 0 at the end of the correct run, and a deleted 1 at the beginning of the correct run.

Based on the above discussion of VT code, we can easily get a one-way single-deletion-correcting synchronization algorithm. In our setting, the length-n string X is at node A, while the node B has string Y, which is obtained by deleting one bit from X. To synchronize, node A only need to send the checksum of X modulo $(n+1)$ to node B. After receiving this value, say a, node B decodes string Y to a codeword in $VT_a(n)$ according to the decoding procedure listed above. We can see that this decoded codeword is equal to X.

During the whole synchronization algorithm, the only message that is needed to sent from node A to node B is $a = \sum_i ix_i \bmod (n+1)$. We call a as the VT syndrome of X. Since $a \in \{0, 1, ..., n\}$, the number of transmitted bits is $\log(n+1)$ bits. This is asymptotically optimal. If Y is obtained from X by a single insertion, an algorithm to synchronize Y to X can be easily derived in a similar fashion. The main difference is that we now use the excess in the checksum of Y (instead of the deficiency) and compare it to its weight. Since we focus on deletion edits in this thesis, the discussion about this single-insertion-correcting synchronization algorithm is omitted here.

3.2.2.2 Synchronizing From Multiple Deletions

Next, we begin to discuss the explicit synchronization algorithm in [12]. We find that the synchronization protocol of [12] can deal with the case of both insertions and deletions. However, the edits that we are concerned with are limited to deletions. Hence, we only give out the algorithm that is designed for deletion-only edits.

The main idea of the synchronization algorithm is to use a divide-and-conquer approach to isolate deletions. Then, the whole file string is broken into many substrings, each of which contains only one deletion. Then we can use the VT syndrome to synchronize these substrings. The algorithm is achieved recursively in the following manner (we reproduce the algorithm procedure from [12]):

1. Node A maintains an unresolved list $\mathcal{L}_X$, whose entries are the yet-to-be-synchronized substrings of X. The list is initialized to be $\mathcal{L}_X = \{X\}$. Node B maintains a corresponding list $\mathcal{L}_Y$, initialized to $\{Y\}$.

2. In each round, node A sends m_a anchor bits around the center of each substring in $\mathcal{L}_X$ to node B, which 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 aligned anchor bits split the substring into two pieces. For each of these pieces:

   - If the number of deletions is *zero*, the piece has been synchronized.
   - If the number of deletions is *one*, node B requests the VT syndrome of this piece for synchronization.
   - If the number of deletions is *greater than one*, node B puts this piece in $\mathcal{L}_Y$. Node A puts its corresponding piece in $\mathcal{L}_X$.

   If one or more of the anchor bits is among the deletions, node B may not be able to align the anchor bits. In this case, in the next round node B requests another set of m_a anchor bits for the substring; this set is chosen adjacent to a previously sent set of anchor bits, as close to the center of the substring as possible. This process continues until node B is able to align a set of anchor bits for that substring.

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

Figure 3.4 illustrates the process of the algorithm. Besides, we add a few more rules to the above synchronization procedure.

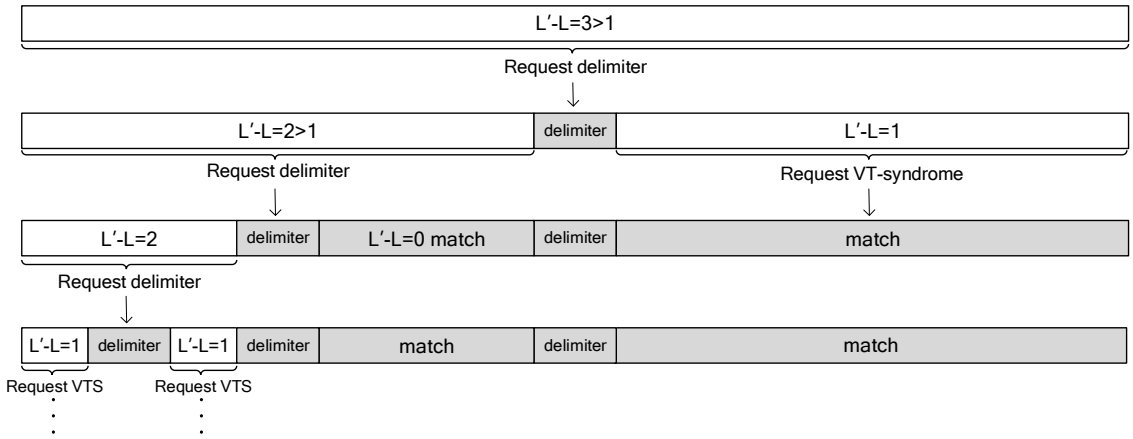

Figure 3.4: Example of a run on the deletion recovery module, in which L' and L respectively denote the length of the string at A and B. At the first iteration (top line), because $L'-L>1$, B requests a central delimiter from A. This delimiter is matched on B, and the algorithm goes on recursively on both sides of the delimiter. Parts of the string that are considered as synchronized are grayed out.

1. When node B receives m_a anchor bits to be aligned within a substring of length l, it searches for a match within a window of length $\kappa\sqrt{l}$ around the middle of its substring, where $\kappa \geq 1$ is a constant.

2. If no matches for the anchor bits are found within this window, node B requests an additional set of anchor bits from a pre-arranged location, chosen as described above.

3. If multiple matches for the anchor are found within the window, node B chooses the match closest to the center of the substring.

4. Whenever an anchor needs to be sent for a piece whose length is less than Lm_a, node B just sends the piece in full. Here $L > 1$ is a pre-specified constant.

5. Whenever the total number of bits transmitted in the course of the algorithm exceeds αn (for some pre-specified $\alpha \in (0, 1)$), we terminate the algorithm and send the entire X sequence.

At last, we list out a theorem, which characterizes the performance of the synchronization algorithm in the deletion recovery module.

Theorem 7 (Venkataramanan, [12]). *Suppose there are d deletions, where $d \sim o(\frac{n}{\log n})$. The positions of the deletions are random and $m_a = c \log n$ anchor 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 $\frac{d \log n}{2n^c}$.

(b) If $N_{A \to B}(d)$ ($N_{B \to A}(d)$) denotes the number of bits transmitted from node A (node B) to node B (node A), then

$$\mathbb{E}N_{A \to B}(d) < (2c+1)d \log n,$$
$$\mathbb{E}N_{B \to A}(d) < 8(d-1).$$

(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 $k+2\log d$ rounds is $2^{-k} + o(2^{-k})$. Consequently, the expected number of rounds taken by the protocol to terminate is approximately $4 + 2 \log d$.

CHAPTER 4

New Protocol

In practice, deletions in files ofter appear in bursts, for example, a paragraph of text is deleted, or several consecutive frames of video are cut. Hence, the i.i.d. deletion pattern assumed in the protocol of Yazdi [14] is not applicable. Since the protocol of Yazdi is optimized for i.i.d. deletions, it might not work very well for the real files. Therefore, it is very significant to utilize the burst nature of deletions in the design of the synchronization algorithm.

In this Chapter, we propose a new synchronization protocol, which can correct bursts of deletions in an efficient way. We build our synchronization algorithm based on the work of Yazdi, and use the same framework with it. Similarly, the new protocol is also consist of three modules: the matching module, the burst deletion recovery module, and the LDPC decoder module. Next, A brief description of the proposed protocol is presented. Then, we will give out more details about the implementation of the matching module and the burst deletion recovery module to get a closer look at our protocol.

4.1 Protocol Overview

First, we take an overview of the new synchronization protocol. The framework of our protocol is the same with the protocol of Yazdi [14], which is shown in Figure 3.1 at Chapter 3. The main difference is that we replace the deletion recovery module with the burst deletion recovery module. This new module adopts a new algorithm to correct the burst deletions within each segment substrings, which is much more efficient than the algorithm [12] used in the protocol of Yazdi.

In a similar fashion, 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 $|S_i| = L_S = 1/d(p_1, p_2)$ and $|P_i| = L_P = O(\log(1/d(p_1, p_2)))$. $d(p_1, p_2)$ is the average deletion rate, which is equal to $(1-p_1)/(2-p_1-p_2)$ from Property 3 of the burst deletion pattern. We choose the length of the segment string $L_S = 1/d(p_1, p_2)$ so that the average number of deletion burst within each segment string is around $1-p_2$, which means that most of segment strings has very few deletion bursts (zero, one or two bursts). This can be derived as follows. From Property 3, we already know that the average length of bursts of non-deleted bits and deleted bits are $1/(1-p_1)$ and $1/(1-p_2)$ respectively. Hence, the average number of deletion burst can be estimated as $L_S/\left(1/(1-p_1) + 1/(1-p_2)\right) = 1-p_2$.

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. After the matching process, we get file Y as

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

We then send back the indices of matched pivots $\{i_1, ..., i_{k'-1}\}$ to node A and get file X as

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

So far, we have gotten many pairs of segment strings $\{(\overline{F}_j, F_j), j = 1, 2, ..., k'\}$. If both $P_{i_{j-1}}$ and P_{i_j} are matched correctly in Y, $\overline{F}_j$ can be derived from F_j by very few bursts of deletions. Then, we need to synchronize $\overline{F}_j$ with the corresponding F_j, which is finished by the burst deletion recovery module. Since the burst nature of deletions destroys the divide-and-conquer approach to isolate deletions, the algorithm of Venkataramanan [12] used in the previous deletion recovery module is no longer efficient. In our burst deletion recovery module, a new synchronization algorithm, which is adapted from the algorithm of Venkataramanan, is used. It is verified that this new synchronization protocol works well when there are very few short bursts of deletions for each segment string [1]. At the end of

[1]From assumption (1) in Chapter 2, we know that the length of every deletion burst is small.

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 with file X. At the last step, we use the LDPC decoder module to correct the residual errors made in the first two modules.

Up to now, we have taken an overview of the new synchronization protocol. Next, we present a theorem in [18], which gives out the minimum rate of any synchronization algorithm under the burst deletion channel modeled by Markov chain.

Theorem 8 (Ma, [18])**.** *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*

$$R_{\min}(p_1, p_2) = -(1-p_1)\log(1-p_1) + (1-p_1)\left(\frac{1 + h_2(1-p_2)}{1-p_2} + \log e - C\right) + O((1-p_1)^{2-\epsilon}), \tag{4.1}$$

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 of that $1 - p_1 \ll 1$ and p_2 is fixed, which coincides with our assumptions given in section 2.2.2. From this theorem, we can get the lower bound of communication rate for our new synchronization protocol. We will show how our new protocol is close to this lower bound in Chapter 5.

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

4.2 Matching Module

Adapting the matching module from the case of i.i.d. deletion pattern in [14] 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. Next, we take a closer look at it.

4.2.1 Correct and Incorrect Matches

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. Based on the number of deletions that acts on each pivot string, we have the following two case (instead of three cases in the i.i.d. deletion pattern):

(1) There is no deletion within P_i. Then, the corresponding copy of P_i in Y is the correct match while all other copies of P_i are incorrect matches.

(2) There is at least one deletion within P_i. Then, all copies of P_i in Y are incorrect matches.

Here, we do not consider the case where exact one deletion occurs in P_i, since the probability of one deletion in P_i is much smaller in burst deletion case compared with i.i.d. deletion case.

Next, Similar to the theoretical analysis about the occurrence of correct matches for P_i, we have the following two Lemmas (for simplicity, we denote the average deletion rate $d(p_1, p_2) = (1-p_1)/(2-p_1-p_2)$ as d hereafter).

Lemma 1. *With probability at least $1 - L_P d + o(d)$, P_i has no deletion and there is one correct match for P_i within Y.*

Proof. For the pivot string of length L_P, the probability of no deletion within the pivot string is $(1-d)p_1^{L_P-1}$. Since $d = \frac{1-p_1}{2-p_1-p_2}$, we have that $1 - p_1 < d$. Also, It can be seen that $L_P = O(\log \frac{1}{d}) \ll \frac{1}{d} < \frac{1}{1-p_1}$. Then, we get that

$$\begin{aligned}
(1-d)p_1^{L_P-1} &= (1-d)(1-(1-p_1))^{L_P-1} \\
&= (1-d)(1-(1-p_1)(L_P-1)+o(1-p_1)) \\
&= 1-(1-p_1)(L_P-1)-d+d(1-p_1)(L_P-1)+o(1-p_1) \\
&> 1-d(L_P-1)-d+o(1-p_1) \\
&= 1-L_Pd+o(d)
\end{aligned}$$

□

Lemma 2. *With probability at most $L_P d + o(d)$, P_i has at least one deletion and there is no correct match for P_i within Y.*

Proof. we can know that the probability of at least one deletion within the pivot string is

$$1 - (1-d)p_1^{L_P - 1} < 1 - (1 - L_P d + o(d)) = L_P d + o(d)$$

□

Similarly, we define $R = 1 - L_P d$. From the preceding lemmas, we conclude that

Theorem 9. *For a random string X and a random burst deletion pattern D defined in Chapter 2, on average, the number of pivots with one correct match in Y is at least $(R + o(d))k$.*

Theorem 10. *For a random string X and a random burst deletion pattern D defined in Chapter 2, with probability $1 - 2^{-\Omega(n)}$, there are at least $(R + o(d))k$ pivots with one correct match in Y.*

The proof of the above two theorems is quite similar to that of the corresponding theorems in [14]. For details, please refer to [14]. From the theorems, we can see that most of pivots has one correct match in Y in the case of burst deletion pattern. Hence, finding a good matching graph to detect these correct matches is very important.

4.2.2 Matching Graph

Note that both our work and the work of Yazdi in [14] focus on the deletion-only edits. Furthermore, we find that the theoretical results about the occurence of correct matches of P_i in the previous section are quite simiar with those in [14]. This inspires us to use the same matching graph to find the correct matches of P_i in the burst deletion case. However, some slight modifications should be made as follows:

- The condition (3.2) is changed as $0 \leq Dis(u, v) \leq (j - i - 1)L_P + (j - i)L_S$. This is due to the small different definition of correct and incorrect matches for P_i.

- We need to find an $s - t$ path of length $(1 - L_P d)k + o(d)k$ in the matching graph (In [14], this length is $(1 - L_P\beta + 2\beta)k + o(\beta)k$).

The numerical experiment shows that, for a proper L_P, all $s - t$ paths with length $(1 - L_P d)k + o(d)k$ is formed mostly of good vertices with a very high probability. However, to get a theoretical guarantee of this, which is similar to Theorem 6, is difficult and beyond the scope of our work. We will discuss the possibility of finishing theoretical work about the matching graph at the end of this thesis.

For details of the construction and the implementation of the matching map, please refer to Section 3.2.1.2 and 3.2.1.3.

4.3 Burst Deletion Recovery Module

After the matching of pivots, we have divided the problem of synchronizing X and Y into many subproblems. for each subproblem, we need to synchronize a segment string in Y to its orignial one in X. Since most of pivots can be correctly matched, we expect that most of segment strings are of length around $1/d$. Furthermore, We already know that most of segment strings in Y can be derived from the corresponding ones in X by very few (usually zero, one or two) short bursts of deletions. The goal of the burst deletion recovery module is to correct these short bursts of deletions within each segment string. Because the divide-and-conquer approach to isolate deletions is very inefficient in the case of burst deletion, we use a new algorithm, which is adapted from the algorithm of Venkataramanan in [12], for the burst deletion recovery module.

Before introduing this new algorithm, we need to describe a method to efficiently synchronizing from a single burst of deletions of known length, which is a key submodule of the new algorithm.

4.3.1 Synchronizing From A Single Deletion Burst

In this section, we make an introduction of a single-deletion-burst synchronization algorithm, which is first proposed by Venkataramanan in [13].

Suppose that Y at node B is obtained from X of length n at node A by deleting a single burst of B bits. In order to get a lower bound on the number of bits required for synchronization, we assume that node A knows the exact location of the burst deletion. Then, the minimal information needed to be sent are the starting position of the burst and the content of deleted bits. Hence, the minimal number of bits required for synchronization can be bounded below by $B+\log n$. Later in this section, We will show that the expected number of transmitted bits for the introduced single-deletion-burst synchronization algorithm is within a small factor of this lower bound.

Now, let us divide each of X and Y into B substrings as follows.

$$X^k = (x_k, x_{B+k}, x_{2B+k}, ...), k = 1, 2, ..., B$$
$$Y^k = (y_k, y_{B+k}, y_{2B+k}, ...), k = 1, 2, ..., B$$

To illustrate the above division of string X and Y, we give out an example.

Example 8. *We have* $X = 10\mathit{011}1001000111$. *After a burst of* $B = 3$ *deletions (shown in italics), we get* $Y = 101001000111$. *Then we have the three substrings of* X *and* Y *as*

$$X^1 = 11001, \quad X^2 = 01001, \quad X^3 = 01101,$$
$$Y^1 = 1001, \quad Y^2 = 0001, \quad Y^3 = 1101.$$

Observe that each of substrings X^k undergoes exactly one deletion to get Y^k. Whenever we have a single burst deletion of B bits, X^k and Y^k differ by exactly one deletion. Furthermore, the positions of the deletions in the substrings X^k, $k = 1, 2, ..., B$ are highly correlated. As we enumerate the substrings X^k from $k = 1$ to $k = B$, the position of the deletion is non-increasing and can decrease at most once. The correlation between positions of deletions suggests a synchronization algorithm as follows.

Node B first synchronize Y^1 to X^1 by receiving the VT syndrome of X^1 from node A, and sends back the position of deletion j back to node A. Then, node A sends the bits in positions $j-1$ and j of X^k, $k = 2, ..., B$, and node B reconstruct each X^k from Y^k, $k = 2, ..., B$ accordingly. This finishes the synchronization of Y to X. In this synchronization algorithm, it assumes that the position of deletion in X^1 can be exactly determined. However, this is not alway possible, since the VT code always inserts a deleted bit either at the beginning or the end of the run containing it.

To address the above issue, we modify the first round of the algorithm as follows. First, node A sends the VT syndromes of both the first and last substring (X^1 and X^B) to node B. Then, node B synchronizes Y^1 and Y^B to X^1 and X^B respectively. It can be known that the deletion in X^1 occurs in the run of positions j_1 to l_1, and the deletion in X^B occurs in the run of position j_B to l_B. From the correlation between positions of deletions given above, we know that the positions of deletions in X^i, $i = 2, 3, ..., B-1$ are between $j^* = \max\{j_1 - 1, j_B\}$ and $l^* = \min\{l_1, l_B + 1\}$. Hence, node A only needs to send the bits in positions j^* to l^* of X^k, $k = 2, 3, ..., B-1$.

Based on the above discussion, we give out the final algorithm for exact synchronization from a single deletion burst of length B as follows.

(1) Node A sends the VT syndrome of X^1 and X^B. (requires $2\log(1 + n/B)$ bits)

(2) Node B synchronizes Y^1 to X^1, and Y^B to X^B. Then node B sends back j^* and l^*, defined as above, to node A. (requires $2\log(n/B)$ bits)

(3) Node A sends bits in positions j^* through l^* of X^k, $k = 2, 3, ..., B-1$. (requires $(l^* - j^* + 1)(B-2)$ bits). And node B reconstruct each X^k from Y^k, $k = 2, ..., B-1$ accordingly.

At last, we present a theorem, which characterizes the expected number of bits required for the above single-deletion-burst synchronization algorithm.

Theorem 11 (Venkataramanan, [13])**.** *Let X be a uniformly random binary sequence of length n. Let Y be obtained via a single burst of deletions of length B, with the starting*

location of the burst being uniformly random. Then for sufficiently large n, the expected number of bits sent by the node A in the synchronization algorithm satisfies

$$\mathbb{E}N_{A\to B} > 2\log(1+n/B) + (2-1/B)(B-2)$$
$$\mathbb{E}N_{A\to B} \leq 2\log(1+n/B) + 3(B-2)$$

The expected number of bits sent by the node B is $2\log(n/(2B))$

The above theorem verifies that the communication rate of the introduced single-deletion-burst synchronization algorithm is within a small factor of the lower bound.

4.3.2 Synchronizing From Few Number of Short Deletion Bursts

Next, we introduce the new algorithm used in the burst deletion recovery module. This algorithm, which is adapted from the algorithm of Venkataramanan in [12] (uses a divide-and-conquer approach to isolate deletions), is very suitable for our case of few number of short deletion bursts.

We still use the divide-and-conquer approach in the new algorithm. However, it is for a different purpose. When a substring with only one short deletion burst is split into two pieces by the anchor bits, we know that one piece has no deletion and the other piece has the same number of deletions with this substring. In contrast, when a substring with multiple short deletion bursts is split into two pieces, it is more likely that both of two pieces contain some deletions. Hence, if a substring is split for T_{burst} (e.g., 2) times and we always have that one piece has no deletion and the other one has the same number of deletions with the original substring, we hypothesize that this substring only contains one deletion burst. Then, we invoke the single-deletion-burst synchronization algorithm in 4.3.1, and use the hash to verify. If the hashes agree, we declare the substring synchronized. Otherwise, we infer that the deletions is not in a burst, and continue to split the substring.

We summary our new algorithm as follows:

1. Node A maintains an unresolved list $\mathcal{L}_X$, whose entries are the yet-to-be-synchronized substrings of X. The list is initialized to be $\mathcal{L}_X = \{X\}$. Node B maintains a corre-

sponding list $\mathcal{L}_Y$, initialized to $\{Y\}$. Besides, we set a counter for each substring in $\mathcal{L}_Y$. Initially, the counter for Y is set as $C_Y = 0$.

2. In each round, node A sends m_a anchor bits around the center of each substring in $\mathcal{L}_X$ to node B, which 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 aligned anchor bits split the substring $str0$ into two pieces $str1$ and $str2$ (We assume that node B and A automatically remove $str0$ and the corresponding substring from the lists if anchor bits are matched). For $str1$ and $str2$:

   - If both of two pieces have some deletions, node B put $str1$ and $str2$ in $\mathcal{L}_Y$. Node A puts the corresponding two pieces in $\mathcal{L}_X$. The counters for these two piece are set as $C_{str1} = C_{str2} = 0$.
   - If one piece $str1$ has no deletion, and the other piece $str2$ has the same number of deletions with $str0$, we set the counter for $str2$ as $C_{str2} = C_{str0} + 1$. Then, if C_{str2} reaches T_{burst}, we invoke the single-deletion-burst synchronization algorithm in 4.3.1, and use the hash to verify. If hashes agree, we declare piece $str2$ synchronized. If hashes do not agree, we set $C_{str2} = 0$. In the case of either ($C_{str2} < T_{burst}$) or ($C_{str2} = T_{burst}$ but hashes do not agree), node B put $str2$ in $\mathcal{L}_Y$, And node A puts the corresponding piece in $\mathcal{L}_X$.

   If one or more of the anchor bits is among the deletions, node B may not be able to align the anchor bits. In this case, in the next round node B requests another set of m_a anchor bits for the substring; this set is chosen adjacent to a previously sent set of anchor bits, as close to the center of the substring as possible. This process continues until node B is able to align a set of anchor bits for that substring.

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

Figure 4.1 illustrates the process of the algorithm. The rules for the synchronization protocol of Venkataramanan [12] are still useful for our new algorithm. For completeness of our algorithm, we reproduce these rules here.

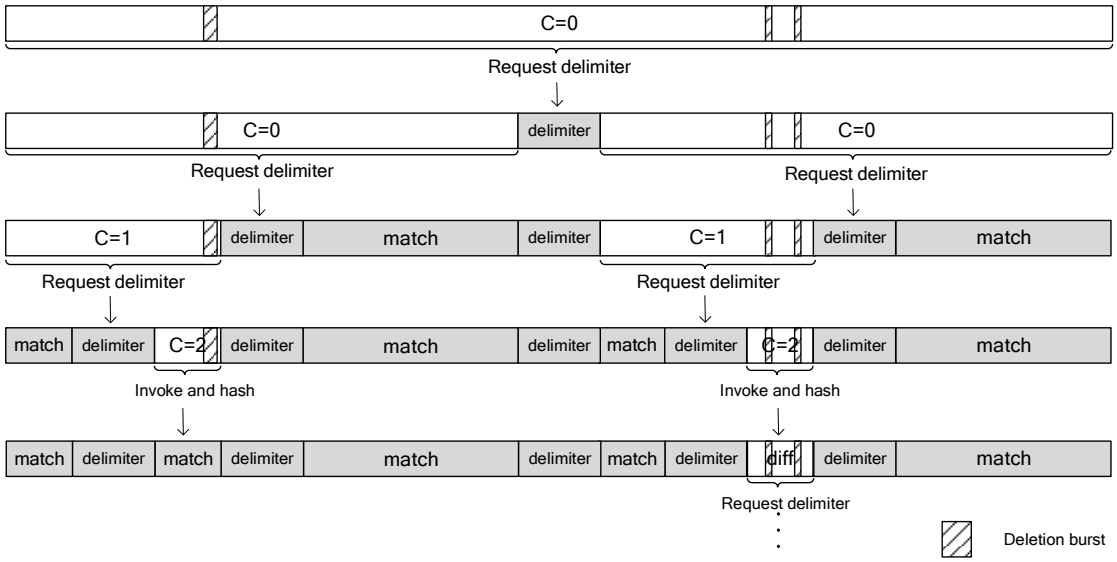

Figure 4.1: Example of a run on the burst deletion recovery module, in which C is the counter for each substring. For the left half part, we have a substring with its counter reaching $T_{burst} = 2$.. We invoke the single-deletion-burst synchronization algorithm and request hash. Since hashes matches, we consider it to be synchronized. For the right half part, there is another substring with its counter reaching T_{burst}. However, due to the mismatch of the hashes, we continue to split this substring. Parts of the string that are considered as synchronized are grayed out.

1. When node B receives m_a anchor bits to be aligned within a substring of length l, it searches for a match within a window of length $\kappa\sqrt{l}$ around the middle of its substring, where $\kappa \geq 1$ is a constant.

2. If no matches for the anchor bits are found within this window, node B requests an additional set of anchor bits from a pre-arranged location, chosen as described above.

3. If multiple matches for the anchor are found within the window, node B chooses the match closest to the center of the substring.

4. Whenever an anchor needs to be sent for a piece whose length is less than Lm_a, node B just sends the piece in full. Here $L > 1$ is a pre-specified constant.

5. Whenever the total number of bits transmitted in the course of the algorithm exceeds αn (for some pre-specified $\alpha \in (0, 1)$), we terminate the algorithm and send the entire X sequence.

CHAPTER 5

Experimental Results

In this Chapter, we report some experimental results about our new protocol, which is specifically designed for the burst deletion case. To determine the relative communication efficiency of the proposed protocol, we compare it with some other synchronization algorithm. In our experiment setting, we use the protocol of Yazdi [14] as the reference algorithm.

Note that the communication efficience involves both the total number of transimitted bits and the number of rounds of interactivity. In some applications where the sources may be connected by a high-latency link, the number of interactive rounds is limited. Hence, we are mostly interested in minimizing the total amount of information transmitted and keeping the number of communication rounds low enough.

5.1 Comparison with the Protocol of Yazdi

Recall that our proposed synchronization protocol is adapted from the protocol of Yazdi. We use the same matching module and adopt a new algorithm in the burst deletion recovery module. Thus, the improvement of our new protocol can be largely attributed to this redesigned new algorithm.

In order to compare the performance of two protocols, the following setup is used for the numerical experiment:

- File X is an i.i.d. Bernoulli(0.5) binary sequence of length $n = 10^6$.
- File Y is derived by applying a burst deletion pattern to file X. This burst deletion pattern is generated by a stationary Markov chain with parameter $(p_1, p_2) = (0.82, 0.9982)$.

	No. of rounds	Total No. of transmitted bits
Our protocol	4	38k
The protocol of Yazdi	9	45k

Table 5.1: Comparison of the protocols of ours and Yazdi. X is an i.i.d. Bernoulli(0.5) binary sequence of length $n = 10^6$. The burst deletion pattern, which is generated by the proposed Markov model with $(p_1, p_2) = (0.82, 0.9982)$, is applied to get Y. The pivots have length 6, and the segments have 100.

- For the matching module of both our protocol and the protocol of Yazdi, the length of segment string and pivot string are set as $L_S = 100$ and $L_P = 6$.
- The number of anchor bits is 4, and the hash length for our protocol is 10 bits.
- T_{burst} is set as 2 for our protocol.
- Each case are averaged over 1000 runs of simulations.

Table 5.1 shows the result. We see that the number of bits required for our protocols is fewer than the protocol of Yazdi. Furthermore, the number of interactive rounds reduces from 9 to 4. The protocol of Yazdi split the strings until each string contains only a single deletion. Hence, the number of rounds is comparable to the logarithm of the segment length. For our new protocol, it invokes the single-deletion-burst synchronization algorithm if we have the case that one substring has no deletion and the other has the same amount of deletions with original string for T_{burst} rounds. Therefore, the saving of rounds for our protocol is roughly $\log L_S - T_{burst}$.

We further notice that the minimum communication rate in (4.1) for $(p_1, p_2) = (0.82, 0.9982)$ is $R_{\min} = 0.0335$. That means that the minimum number of transmitted bits is $nR_{\min} = 33.5k$. So the communication rate of our protocol is close to this lower bound.

CHAPTER 6

Conclusion

6.1 Summary of Our Work

In this thesis, we proposed a new synchronization protocol, which is specifically designed for the burst deletion case. In order to model this burst deletion pattern, we adopt the stationary two-state Markov chain model with the appropriately seleted parameters. And the experiment results demonstrate that our new protocol has lower communication rate and fewer rounds of interactivity.

Nowadays, there is a large amount of research work about the synchronization problem. However, very few of them is focused on burst edits. Some recently proposed synchronization algorithms assume that the edits is independent and uniformly distributed on the whole file string. But in practical scenarios, this assumption does not hold in general. Burst edits is more often. Hence, those protocols optimized for non-burst edits might be inefficient when they are used for read edit pattern. Our proposed synchronization protocol make use of the burst nature of edits (we limit edits to deletions in the thesis), and hence works well for burst deletion case. This new protocol is gotten based on the work of Yazdi. The framework of the proposed protocol is same with the protocol of Yazdi while some submodules within the protocol are accordingly adjusted to be more suitable for the burst deletion correction.

In addition, to make the burst deletion pattern mathematically tractable, we propose to use the stationary two-state Markov chain to model it. We give out many useful properties of the Markov chain, and show that the desired burst deletion pattern can be generated by choosing appropriate parameters. Also, some simplifying assumptions are made to make our protocol more efficient and some relecant mathematical analysis more tractable.

6.2 Future Work

One important extension of our work is the design of synchronization protocol that are capable of recovering from both burst deletions and insertions. In order to model burst deletions and insertions, we need to use three-state Markov chain model defined as follows.

$$T = \begin{bmatrix} p_1 & 1-p_2 & 1-p_2 \\ \frac{1-p_1}{2} & p_2 & 0 \\ \frac{1-p_1}{2} & 0 & p_2 \end{bmatrix},$$

in which $p_1 = P(D_i = 0|D_{i-1} = 0)$, $p_2 = P(D_i = 1|D_{i-1} = 1) = P(D_i = 2|D_{i-1} = 2)$. $D_i = 0, 1, 2$ means that the ith bit is kept, deleted or inserted respectively. Some similar assumptions can be made to simplify the synchronization problem. However, to adapt the matching module to the case of burst deletions and insertions is somewhat challenging.

Based on the proposed protocol, some theoretical work can be done to guarantee the communication efficiency of our protocol. From the theoretical results that is finished now (see section 4.2.1), we find that the average deletion rate $d = (1-p_1)/(1-p_1-p_2)$ plays the same role as β does in some theorems. Hence, many theorems in the i.i.d. deletion pattern may be generalized to the case of burst deletion pattern by replacing β with d. However, whether this is feasible need more mathematical proof, which should be very complicated.

In some cases, we might not need exact synchronization. Hence, the design of the approximate synchronization algorithm based our protocol is also meaningful. By allowing the reconstructed file a bit different from the original file, the communication rate of the protocol is expected to be smaller.

REFERENCES

[1] M. Mitzenmacher and G. Varghese, "The complexity of object reconciliation, and open problems related to set difference and coding," in *Proc. IEEE 50th Allerton Conf. Commun., Control, Comput.*, Monticello, IL, USA, Oct. 2012, pp. 1126-1132.

[2] R. R. Varshamov and G. M. Tenengolts, "Codes which correct single asymmetric errors," *Autom. Remote Control*, vol. 26, no. 2, pp. 288-292, 1965.

[3] 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.

[4] G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion (Corresp.)," *IEEE Trans. Inf. Theory*, vol. IT-30, no. 5, pp. 766-769, Sep. 1984.

[5] A. S. J. Helberg and H. C. Ferreira, "On multiple insertion/deletion correcting codes," *IEEE Trans. Inf. Theory*, vol. 48, no. 1, pp. 305-308, Jan. 2002.

[6] K. A. S. Abdel-Ghaffar, F. Paluncic, H. C. Ferreira, and W. A. Clarke, "On Helbergs generalization of the Levenshtein code for multiple deletion/insertion error correction," *IEEE Trans. Inf. Theory*, vol. 58, no. 3, pp. 1804-1808, Mar. 2012.

[7] L. Dolecek and V. Anantharam, "Repetition error correcting sets: Explicit constructions and prefixing methods," *SIAM J. Discrete Math.*, vol. 23, no. 4, pp. 2120-2146, Jan. 2010.

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

[9] 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.

[10] 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.

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

[12] R. Venkataramanan, H. Zhang, and K. Ramchandran, "Interactive lowcomplexity 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] H. Zhang, C. Yeo, and K. Ramchandran, "VSYNC: A novel video file synchronization protocol, in *Proc. 16th ACM Int. Conf. Multimedia*, Vancouver, BC, Canada, Oct. 2008, pp. 757-760.

[21] B. Brainerd and S. M. Chang, "Number of occurrences in two-state Markov chains, with an application in linguistics," *Canad. J. Statist.*, vol. 10, no. 3, pp. 225-231, 1982.

[22] B. M. McCoy and T. T. Wu. "The two-dimensional Ising model." Harvard Univ. Press, 1973.