Coding for Segmented Edit Channels

Mahed Abroshan, Ramji Venkataramanan and Albert Guillén i Fàbregas

Abstract

We consider insertion and deletion channels with the additional assumption that the channel input sequence is implicitly divided into segments such that at most one edit can occur within a segment. No segment markers are available in the received sequence. We propose code constructions for the segmented deletion, segmented insertion, and segmented insertion-deletion channels based on subsets of VT codes chosen with pre-determined prefixes and/or suffixes. The proposed codes, constructed for any finite alphabet, are zero-error and can be decoded segment-by-segment. We also derive an upper bound on the rate of any zero-error code for the segmented edit channel, in terms of the segment length. This upper bound shows that the rate scaling of the proposed codes as the segment length increases is the same as that of the maximal code.

I. INTRODUCTION

We consider the problem of constructing codes for segmented edit channels, where the channel input sequence is implicitly divided into disjoint segments. Each segment can undergo at most one edit, which can be either an insertion or a deletion. There are no segment markers in the received sequence.

This model, introduced by Liu and Mitzenmacher [1], is a simplified version of the general edit channel, where the insertions and deletions can be arbitrarily located in the input sequence. Constructing codes for general edit channels is well known to be challenging problem; see, e.g., [2]–[9]. The assumption of segmented edits not only simplifies the coding problem, but is also
likely to hold in many edit channels that arise in practice, e.g., in data storage and in sequenced genomic data, where the number of edits is small compared to the length of the input sequence. As explained in [1], when edits (deletions or insertions of symbols) occur due to timing mismatch between the data layout and the data-reading mechanism, there is often a minimum gap between successive edits. The segmented edit model includes such cases, though it also allows for nearby edits that cross a segment boundary. Furthermore, a complete understanding of the segmented edit model may provide insights into the open problem of constructing efficient, high-rate codes for general edit channels. As we show in this paper, the segmented edit assumption allows for the construction of low-complexity, zero-error codes with the optimal rate scaling for any finite alphabet.

Let us consider three examples to illustrate the model. For simplicity, we consider a binary alphabet and assume that the segment length, denoted by $b$, is 3 in each case.

1) **Segmented Deletion Channel:** Each segment can undergo at most one deletion; no insertions occur. Consider the following pair of input and output sequences:

$$X = 011\underline{1}00\underline{0}10 \rightarrow Y = 0110010,$$

with the underlined bits in $X$ being deleted by the channel to produce the output sequence $Y$. It is easily verified that many other input sequences could have produced the same output sequence, e.g., $010\underline{1}00\underline{0}10, 010\underline{1}01\underline{0}10, 011\underline{0}00\underline{1}00$ etc. The receiver has no way of distinguishing between these candidate input sequences. In particular, despite knowing the segment length and that deletions occurred, it does not know in which two segments the deletions occurred.

2) **Segmented Insertion Channel:** Each segment can undergo at most one insertion; no deletions occur. The inserted bit can be placed anywhere within the segment, including before the first bit or after the last bit of the segment. For example, consider

$$X = 011\underline{1}00\underline{0}10 \rightarrow Y = 011\underline{1}0\underline{1}001\underline{1}0,$$

with the underlined bits in $Y$ indicating the insertions. Two inserted bits can appear between two segments whenever there is an insertion after the last bit of first segment and before the first bit of the next segment.

3) **Segmented Insertion-Deletion Channel:** This is the most general case, where a segment could undergo either an insertion or a deletion, or remain unaffected. For example, consider

$$X = 011\underline{1}00\underline{0}10 \rightarrow Y = 01\underline{0}10001\underline{1}0,$$
with the underlined bits on the left indicating deletions, and the underlined bits on the right indicating insertions. Unlike the previous two cases, the receiver cannot even infer the exact number of edits that have occurred. In the example above, an input sequence 9 bits (three segments) long could result in a 10-bit output sequence in two different ways: either via one segment with an insertion, or via two segments with insertions and the other with a deletion.

The above examples demonstrate that one cannot reduce the problem to one of correcting one edit in a $b$-bit input sequence. To see this, consider the example in (1), and suppose that we used a single-deletion correcting code for each segment. Such a code would declare the first three bits of $Y$ to be the first segment of $X$, which would result in incorrect decoding of the following segments.

In this paper, we construct zero-error codes for each of the three segmented edit models above, for any finite alphabet of size $q \geq 2$. Our codes can easily be constructed even for relatively large segment sizes (several tens), and can be decoded segment-by-segment in linear time. Moreover, the proposed codes have rate $R$ of at least

$$R \geq \log_2 q - \frac{1}{b} \log_2 (b+1) - \frac{\kappa}{b} \log_2 q,$$

where the constant $\kappa$ is at most 2.5 for the segmented deletion channel, 4 for the segmented insertion channel, and 8 for the segmented insertion-deletion channel. (Slightly better bounds on $\kappa$ are obtained for the binary case $q = 2$.)

We also derive an upper bound in terms of the segment length $b$ on the maximum rate of any code for the segmented edit channel. This upper bound (Theorem 1) shows that the rate $R$ of any zero-error code with code length $n$ satisfies

$$R \leq \log_2 q - \frac{1}{b} \log_2 b - \frac{1}{b} \log_2 (q-1) + \frac{\log_2 (2q)}{n} + O \left( \frac{\ln b}{b^{4/3}} \right).$$

Comparing (4) and (5), we see that the rate scaling for the proposed codes is the same as that of the maximal code with the rate penalty being $O(1/b)$.

The starting point for our code constructions is the family of Varshamov-Tenengolts (VT) codes [2], [10], [11]. Each code in this family is a single-edit correcting code. In our constructions, the codewords in each segment are drawn from subsets of VT codes satisfying certain prefix/suffix conditions, which are carefully chosen to enable fast segment-by-segment VT decoding.
A. Comparison with previous work

The segmented edit assumption places a restriction on the kinds of edit patterns that can be introduced in the input sequence. Other models with restrictions on edit patterns include the forbidden symbol model considered in [12].

We now highlight some similarities and differences from the codes proposed by Liu and Mitzenmacher in [1] for the binary segmented deletion and segmented insertion channels.

**Code construction:** The code in [1] is a binary segment-by-segment code specified via sufficient conditions [1, Theorems 2.1, 2.2] that ensure that as decoding proceeds, there are at most two choices for the starting position of the next undecoded segment. Finding the maximal code that satisfies these conditions corresponds to an independent set problem, which is challenging for large $b$. The maximal code satisfying these conditions was reported in [1] for $b = 8, 9$. For larger $b$, a greedy algorithm was used to find a set of codewords satisfying the conditions. It was also suggested that one could restrict the code to a subset of VT codes that satisfy the sufficient conditions.

In comparison, our codes are directly defined as subsets of VT codes that satisfy certain simple prefix/suffix conditions; these conditions are different from those in [1]. Our conditions ensure that upon decoding each segment, there is no ambiguity in the starting position of the next segment. These subsets of VT codes are relatively simple to enumerate, so it is possible to find the largest code satisfying our conditions for $b$ of the order of several tens. Table I lists the number of codewords per segment for the three segmented edit channels for $q = 2$ and lengths up to $b = 24$. For the segmented deletion and segmented insertion-deletion channels, another difference from the code in [1] is that our codebook for each segment is chosen based on the final bit of the previous segment.

**Rate**: The VT subsets and sufficient conditions we define allow us to obtain a lower bound of the form (4) on the rate of our code for any segment length $b$. Though the maximal codes satisfying the Liu-Mitzenmacher conditions have rate very close to the largest possible with segment-by-segment decoding, finding the maximal code satisfying these conditions is computationally hard, so one has to resort to greedy algorithms to construct codes for larger $b$. This is reflected in the rate comparison: for $b = 8, 9$, the optimal Liu-Mitzenmacher code for segmented deletions is larger than our code (12,20 vs. 8,13 codewords). However for $b = 16$, the code obtained in [1] using a greedy algorithm has 652 codewords, whereas our code has 964
codewords, as shown in Table I. For large $b$, our codes are nearly optimal since the rate penalty decays as $κ/b$.

For the segmented insertion channel, it is shown in Sec. V-C that our code construction satisfies the sufficient conditions specified [1]. The lower bound on the rate of our code affirmatively answers the conjecture in [1] that the rates of the maximal codes satisfying the sufficient conditions increases with $b$.

**Encoding and decoding complexity:** Being subsets of VT codes, our codes can also be efficiently encoded even for large segment sizes $b$, without the need for look-up tables [13]. As segment-by-segment decoding is enforced by design, the decoding complexity grows linearly with the number of segments for both our codes and those in [1]. Within each segment, the decoding complexity of our code is also linear in $b$, since VT codes can be decoded with linear complexity [2]. In general, for each segment, the maximal Liu-Mitzenmacher codes have to be decoded via look-up tables, in which case the complexity is exponential in $b$. Using subsets of VT codes was suggested in [1] as a way to reduce the decoding complexity.

Finally, we remark that codes proposed in this paper are the first for the binary segmented insertion-deletion model, and for all the non-binary segmented edit models.

**B. Organization of the paper**

The remainder of the paper is organized as follows. In Section II, we formally define the channel model, and review binary and non-binary VT codes. In Section III, we derive an upper bound on the rate of any code for a segmented edit channel, in terms of the segment length. In Sections IV, V, and VI we present our code constructions for the segmented deletion channel, segmented insertion channel, and the segmented insertion-deletion channel, respectively. For each model, we first treat the binary case to highlight the key ideas, and then extend the construction to general non-binary alphabets.

**II. CHANNEL MODEL AND PRELIMINARIES**

The channel input sequence is denoted by $X = x_1x_2 \cdots x_n$, with $x_i \in \mathcal{X}$ for $i = 1, \ldots, n$, where $\mathcal{X} = \{0, \ldots, q - 1\}$ is the input alphabet, with $q \geq 2$. The channel input sequence is divided into $k$ segments of $b$ symbols each. We denote the subsequence of $X$, from index $i$ to index $j$, with $i < j$ by $X(i : j) = x_i x_{i+1} \cdots x_j$. The $i$-th segment of $X$ is denoted by $S_i = s_{i,1} \cdots s_{i,b} = X(b(i - 1) + 1 : bi)$ for $i = 1, \ldots, k$. 
In the segmented deletion channel, the channel output \( Y = Y(1 : m) = y_1 \cdots y_m \), with \( m \leq n \) is obtained by deleting at most one symbol in each segment, i.e., at most one symbol in \( S_i, i = 1, \ldots, k \), is deleted. Similarly, in the segmented insertion channel, the channel output \( Y = y_1 \cdots y_m \), with \( m \geq n \) is obtained by inserting at most one symbol per segment. In the segmented insertion-deletion channel, the channel output is such that each segment \( S_i, i = 1, \ldots, k \) undergoes at most one edit. In all cases, we assume that the decoder knows \( k \) and \( b \), but not the segment boundaries.

We consider coded communication using a code \( \mathcal{C} = \{X^{(1)}, \ldots, X^{(M)}\} \subseteq \mathcal{X}^n \) of length \( n \), \( M \) codewords and rate \( R = \frac{1}{n} \log_2 M \). We consider segment-by-segment coding, where \( M_s \) is the number of codewords per segment. The overall code of length \( n = kb \) has \( (M_s)^k \) codewords, and rate

\[
R = \frac{1}{n} \log_2 (M_s)^k
\]

\[
= \frac{1}{b} \log_2 M_s.
\]
The decoder produces an estimate \( \hat{X} \) of the transmitted sequence. We denote the corresponding segment estimates by \( \hat{S}_i = \hat{s}_{i,1} \cdots \hat{s}_{i,n_i} \), for \( i = 1, \ldots, k \). Thus \( \hat{X} = (\hat{S}_1, \ldots, \hat{S}_k) \). We consider zero-error codes that always ensure the recoverability of the transmitted sequence, i.e., codes for which \( \hat{X} = X \).

A. Binary VT codes

First consider the case where \( q = 2 \), i.e., \( \mathcal{X} = \{0,1\} \). Suppose that \( k = 1 \), and thus \( n = b \), i.e., there is at most one edit in the entire binary sequence. For this model, one can use binary VT codes which are zero-error single-edit correcting codes \[2\], \[10\], i.e., when the transmitted codeword suffers a single insertion or a deletion, the decoder always corrects the edit. Moreover, the complexity of the VT decoding algorithm is linear in the code length \( b \). The details of the decoding algorithm can be found in \[2\] for the case of a single deletion; the decoding algorithm to correct from a single insertion can be found in \[11\], Sec. II.

The VT syndrome of a binary sequence \( S = s_1 \ldots s_b \) is defined as

\[
\text{syn}(S) = \sum_{j=1}^{b} j s_j \mod (b+1).
\] (8)

For positive integers \( b \) and \( 0 \leq a \leq b \), we define the VT code of length \( b \) and syndrome \( a \), denoted by

\[
\mathcal{V}T_a(b) = \{ S \in \{0,1\}^b : \text{syn}(S) = a \}
\] (9)

i.e., the set of sequences \( S \) of length \( b \) that satisfy \( \text{syn}(S) = a \). For example,

\[
\mathcal{V}T_1(3) = \{ s_1s_2s_3 : \sum_{j=1}^{3} j s_j = 1 \mod 4 \} = \{100, 011\}.
\] (10)

The \( b + 1 \) sets \( \mathcal{V}T_a(b) \subset \{0,1\}^b \), for \( 0 \leq a \leq b \), partition the set of all sequences of length \( b \). Each of these sets \( \mathcal{V}T_a(b) \) is a single-edit correcting code. Moreover, for \( 0 \leq a \leq b \), the cardinalities of these sets satisfy \[2\] Corollary 2.3

\[
|\mathcal{V}T_0(b)| \geq |\mathcal{V}T_a(b)| \geq |\mathcal{V}T_1(b)|.
\] (11)

The largest of the sets \( \mathcal{V}T_a(b) \), \( 0 \leq a \leq b \), will have at least \( \frac{2^b}{b+1} \) sequences out of the \( 2^b \) possible. This induces a rate \( R \geq 1 - \frac{1}{b} \log_2(b+1) \) for the largest of these codes. The code \( \mathcal{V}T_0(b) \) has been shown to be maximal for single edit correction for \( b \leq 8 \), and has been conjectured to be maximal for arbitrary \( b \) \[2\].
B. Non-binary VT codes

Here we consider the case where \( X = \{0, \ldots, q-1\} \), with \( q > 2 \). Again, suppose that \( k = 1 \) and thus \( n = b \), i.e., there is at most one edit in the sequence. For this model, one can use \( q \)-ary VT codes, introduced by Tenengolts [11]. These are zero-error single-edit correcting codes, analogous to the binary VT codes. We briefly describe the code construction below.

For each non-binary sequence \( S \), define a length \((b - 1)\) auxiliary binary sequence \( A_S = \alpha_2, \ldots, \alpha_b \) as follows. For \( 2 \leq i \leq b \),

\[
\alpha_i = \begin{cases} 
1 & \text{if } s_i \geq s_{i-1} \\
0 & \text{if } s_i < s_{i-1} 
\end{cases}
\]  

We also define the modular sum as

\[
\text{sum}(X) = \sum_{i=1}^{b} s_i \pmod{q}.
\]

The \( q \)-ary VT code with length \( b \) and parameters \((a, c)\) is defined as [11]

\[
\mathcal{VT}_{a,c}(b) = \{ S \in X^b : \text{syn}(A_S) = a, \text{sum}(S) = c \},
\]

for \( 0 \leq a \leq b-1 \) and \( c \in X \). Similarly to the binary case, the sets \( \mathcal{VT}_{a,c}(b) \) for \( 0 \leq a \leq b-1 \) and \( c \in X \) partition the space \( X^b \) of all \( q \)-ary sequences of length \( b \). Clearly, the largest codebook has at least \( \frac{q^b}{b^b} \) codewords which implies the following rate lower bound for the largest VT code among all choices of \((a,c)\):

\[
R \geq \log_2 q - \frac{1}{b} \log_2 b - \frac{1}{b} \log_2 q.
\]

The complexity of the decoding algorithm for \( q \)-ary VT codes is linear in the code length \( b \). The details of the decoder can be found in [11, Sec. II].

III. UPPER BOUND ON RATE

In this section, we derive an upper bound on the rate of any code for \( q \)-ary segmented edit channels, for \( q \geq 2 \). The upper bound is valid for all zero-error codes, including those that cannot be decoded segment-by-segment.

**Theorem 1.** For each of the three segmented edit models, with segment length \( b \), the rate \( R \) of any zero-error code with code length \( n = kb \) satisfies

\[
R \leq \log_2 q - \frac{1}{b} \log_2 b - \frac{1}{b} \log_2 (q - 1) + \frac{\log_2 (2q)}{kb} + O \left( \frac{\ln b}{b^{4/3}} \right).
\]

(16)
Remarks:

1) In the theorem, the alphabet size $q$ is held fixed as the segment size $b$ grows. The number of segments per codeword, $k$, is arbitrary, and need not grow with $b$.

2) The theorem is obtained via non-asymptotic bounds on the size and the rate of any zero-error code. These bounds, given in (37)–(41), may be of independent interest.

Proof of Theorem 1: We give the proof for the segmented deletion model with segment length $b$. The argument for the segmented insertion model is similar.

The proof technique is similar to that used by Tenengolts in [11, Theorem 2]. The high-level idea is the following. The codewords are split into two groups: the first group contains the codewords in which a large majority of segments have at least $b\frac{(q-1)}{q} - O(b^{2/3})$ runs. The other group contains the remaining codewords. As $b$ grows larger, the fraction of length $b$ sequences with close to $b\frac{(q-1)}{q}$ runs (the ‘typical’ value) approaches 1. So we carefully bound the number of codewords in the first group, while the number of codewords in the second group can be bounded by a direct counting argument.

Consider a code $C$ of length $n = kb$, i.e., each codeword has $k$ segments of length $b$. Let $M = |C| = 2^{nR}$ denote the size of the code. For integers $r \geq 0$ and $0 \leq l \leq k$, define $\mathcal{M}(r, l) \subset C$ as the set of the codewords that have exactly $l$ segments with more than $r$ runs. Let $M(r, l) = |\mathcal{M}(r, l)|$. Note that for any $r \geq 0$, we have

$$
\sum_{l=0}^{k} M(r, l) = M. \quad (17)
$$

For any $l \leq k$ and a codeword $x \in \mathcal{M}(r, l)$, let $\rho_l(x)$ denote the number of distinct sequences of length $(n - l)$ by deleting exactly $l$ symbols from $x$ (following the segmented assumption). We then have

$$(r - 1)^l \leq \rho_l(x). \quad (18)$$

To show (18), we only need to consider $r \geq 3$ as the inequality is trivial for $r \leq 2$. Considering the $l$ segments that each have at least $(r + 1)$ runs, there are at least $(r - 1)^l$ ways of choosing one run from each segment so that the $l$ chosen runs are non-adjacent. For each such choice of $l$ non-adjacent runs, we get a distinct subsequence of length $(n - l)$ by deleting one symbol from each run. This proves (18).
Since $C$ is a zero-error code, for two distinct codewords $x_1, x_2 \in M(r, l)$, the set of length $(n - l)$ sequences obtained via $l$ deletions (in a segmented manner) from $x_1$ must be distinct from the corresponding set for codeword $x_2$. We therefore have

$$q^{n-l} \geq \sum_{x \in M(r, l)} \rho_l(x)$$  \hspace{1cm} (19)

$$\geq \sum_{x \in M(r, l)} (r-1)^l$$  \hspace{1cm} (20)

$$= M(r, l)(r-1)^l,$$  \hspace{1cm} (21)

where $(a)$ is obtained from (18). We therefore obtain

$$M(r, l) \leq \frac{q^{n-l}}{(r-1)^l}. \hspace{1cm} (22)$$

Fix $\alpha \in (0, 1)$. Summing (22) over $\alpha k \leq l \leq k$, we obtain

$$\sum_{l \geq \alpha k} M(r, l) \leq \sum_{l \geq \alpha k} \frac{q^{n-l}}{(r-1)^l}$$  \hspace{1cm} (23)

$$\leq \frac{2q^{n-\alpha k}}{(r-1)^{\alpha k}}. \hspace{1cm} (24)$$

Now choose

$$r = \frac{(q-1)b}{q} - \sqrt{\frac{2\kappa(q-1)b \ln b}{q}}, \hspace{1cm} (25)$$

where $\kappa > \frac{\ln(2q)}{\ln q}$ will be specified later. Using this $r$ in (22), and noting that $n = kb$, we have

$$\sum_{l \geq \alpha k} M(r, l) \leq \frac{2q^{kb-\alpha k}}{(r-1)^{\alpha k}} \hspace{1cm} (26)$$

$$= \frac{2q^{kb}}{(b(q-1))^{\alpha k} \left(1 - \sqrt{\frac{2eq \ln b}{(q-1)b - \frac{q}{(q-1)b}}} \right)^{\alpha k}. \hspace{1cm} (27)$$

For $l < \alpha k$, we use the looser bound

$$M(r, l) \leq \left[q \sum_{t=0}^{r-1} (q-1)^t \binom{b-1}{t} \right]^{k-l} q^{bl}, \hspace{1cm} (28)$$

which is obtained as follows. A segment with $t$ runs is determined by the choice of the first symbol, and the starting positions and values of the next $(t-1)$ runs. There are $q$ choices for the first symbol, $(\binom{b-1}{t-1})$ choices for the starting position of the next $(t-1)$ runs, and $(q-1)^{t-1}$ choices for the values of these runs. Therefore, the number of possible length $b$ sequences with at most $r$ runs is $q \sum_{t=1}^{r} (\binom{b-1}{t-1}) (q-1)^{t-1} = q \sum_{t=0}^{r-1} (\binom{b-1}{t}) (q-1)^t$. We then obtain (28) by noting
that: i) there are \((k - l)\) segments with at most \(r\) runs, and ii) there are at most \(q^l\) choices for the remaining \(l\) segments.

We write the right hand side of (28) as
\[
\left[ q \sum_{t=0}^{r-1} (q-1)^t \binom{b-1}{t} \right]^{k-l} q^l = \left[ q^{b+1} \sum_{t=0}^{r-1} \left( \frac{1}{q} \right)^t \left( \frac{1}{q} \right)^{b-t} \right]^{k-l} q^l = q^{bk+k-l} \left[ \sum_{t=0}^{r-1} \left( \frac{1}{q} \right)^t \left( \frac{1}{q} \right)^{b-t} \left( \frac{b-1}{t} \right) \right]^{k-l} .
\]

It is shown in Appendix A that
\[
\sum_{t=0}^{r-1} \left( \frac{1}{q} \right)^t \left( \frac{1}{q} \right)^{b-t} \left( \frac{b-1}{t} \right) \leq \frac{1}{br^c} .
\]

Using (31) to bound (30), and then substituting in (28), we obtain
\[
M(r, l) \leq \frac{q^{bk+k-l}}{b^{c(k-l)}} .
\]

Summing over \(0 \leq l < \alpha k\) and considering \(\kappa > \frac{\log(2q)}{\log b}\), we obtain
\[
\sum_{l<\alpha k} M(r, l) \leq \frac{q^{(b+1)k}}{b^{c(1-\alpha)k}} \sum_{l<\alpha k} \left( \frac{b^\kappa}{q} \right)^l \leq \frac{q^{(b+1-\alpha)k+1}}{b^{c(1-\alpha)k}} .
\]

Combining the bounds in (27) and (34), we have
\[
M = \sum_{l=0}^{k} M(r, l) \leq \frac{2 q^{kb}}{(b(q-1))^\alpha k} \left( 1 - \sqrt{\frac{2q \ln b}{(q-1)b} - \frac{q}{(q-1)b}} \right)^\alpha k + \frac{q^{(b+1-\alpha)k}}{b^{c(1-\alpha)k}} \leq 2 \max\{T_1, T_2\} .
\]

where
\[
T_1 = \frac{2 q^{kb}}{(b(q-1))^\alpha k} \left( 1 - \sqrt{\frac{2q \ln b}{(q-1)b} - \frac{q}{(q-1)b}} \right)^\alpha k, \quad T_2 = \frac{q^{(b+1-\alpha)k+1}}{b^{c(1-\alpha)k}} .
\]

Therefore the rate can be bounded as
\[
R = \frac{\log M}{kb} \leq \frac{1}{kb} + \max \left\{ \frac{\log T_1}{kb}, \frac{\log T_2}{kb} \right\} .
\]
From (38), we have

\[
\frac{\log T_1}{kb} \leq \log_2 q - \frac{\alpha \log_2 (b(q-1))}{b} + \frac{\alpha}{b} \log_2 \left( 1 - \sqrt{\frac{2\kappa q \ln b}{(q-1)b}} - \frac{q}{(q-1)b} \right) + \frac{\log_2 q}{kb}, \tag{40}
\]

\[
\frac{\log T_2}{kb} \leq \log_2 q - \frac{\kappa (1 - \alpha) \log_2 b}{b} + \frac{(1 - \alpha) \log_2 q}{b} + \frac{\log_2 q}{kb}. \tag{41}
\]

Now choose \(\alpha\) and \(\kappa\) as follows:

\[
\alpha = 1 - \frac{1}{\sqrt{b}}, \tag{42}
\]

\[
\kappa = \frac{\alpha \log_2 (b(q-1))}{1 - \alpha}, \tag{43}
\]

\[
= \left( \sqrt{b} - 1 \right) \frac{\log_2 (b(q-1))}{\log_2 b}. \tag{44}
\]

Note that we have \(\alpha \to 1\) and \(\frac{2\kappa q \ln b}{(q-1)b} \to 0\) as \(b \to \infty\). Using the fact that \(\ln(1/(1-x)) \leq 2x\) for \(x \in (0, 1/2]\) in (40), we have the following bound on \(T_1\) for sufficiently large \(b\):

\[
\frac{\log T_1}{kb} \leq \log_2 q - \frac{\alpha \log_2 (b(q-1))}{b} + \frac{\log_2 q}{kb} + \frac{2\alpha}{b \ln 2} \left( \sqrt{\frac{2\kappa q \ln b}{(q-1)b}} + \frac{q}{(q-1)b} \right)
\]

\[
= \log_2 q - \frac{\log_2 (b(q-1))}{b} + \frac{\log_2 (b(q-1))}{b^{4/3}} + \frac{\log_2 q}{kb} + \frac{2\alpha}{b \ln 2} \left( \sqrt{\frac{2\kappa q \ln b}{(q-1)b}} + \frac{q}{(q-1)b} \right). \tag{45}
\]

Also substituting the values of \(\alpha, \kappa\) from (42) and (44) in (41), we have

\[
\frac{\log T_2}{kb} \leq \log_2 q - \frac{\log_2 (b(q-1))}{b} + \frac{\log_2 (b(q-1))}{b^{4/3}} + \frac{\log_2 q}{kb} + \frac{\log_2 q}{kb}. \tag{46}
\]

Finally, substituting the values of \(\alpha, \kappa\) into the last term in (45), it can be seen that this term is \(O(\sqrt{\ln b/b^{4/3}})\), which yields the desired result.

**IV. SEGMENTED DELETION CODES**

In this section, we show how to construct a segment-by-segment zero-error code for the segmented deletion channel. For simplicity, we first introduce binary codes and explain the binary decoder. We then highlight the differences in the non-binary case.

If the decoder knew the segment boundaries, then simply using a VT code for each segment would suffice. Since the segment boundaries are not known, recall from the example in (1) that this approach is inadequate if segment-by-segment decoding is to be used. Our construction chooses a subset of a VT code for each segment, with prefixes determined by the last symbol of the previous segment.
A. Binary Code Construction

For \(0 \leq a \leq b\), define the following sets.

\[
A^0_a \triangleq \{ S \in \{0, 1\}^b : \text{syn}(S) = a, \ s_1s_2 = 00 \},
\]

\[
A^1_a \triangleq \{ S \in \{0, 1\}^b : \text{syn}(S) = a, \ s_1s_2 = 11 \}.
\]

(47)

For \(c \in \{0, 1\}\), the set \(A^c_a \subseteq \\mathcal{V}_a(b)\) is the set of VT codewords that start with prefix \(cc\). We now choose the sets with the largest number of codewords, i.e., we choose \(A^{0}_{a_0}\) and \(A^{1}_{a_1}\) where we define

\[
a_0 = \arg \max_{0 \leq a \leq b} |A^0_a|, \quad a_1 = \arg \max_{0 \leq a \leq b} |A^1_a|.
\]

(48)

By defining \(M_s = \min\{|A^{0}_{a_0}|, |A^{1}_{a_1}|\}\), we can now construct \(A^0 \subseteq A^{0}_{a_0}\) by choosing any \(M_s\) sequences from \(A^{0}_{a_0}\); similarly construct \(A^1 \subseteq A^{1}_{a_1}\) by choosing any \(M_s\) sequences from \(A^{1}_{a_1}\). The sets \(A^0\) and \(A^1\) are subsets of the VT codes \(\mathcal{V}_a(b)\) and \(\mathcal{V}_a(b)\), containing sequences starting with 00 and 11, respectively.

Finally, the overall code of length \(n = kb\) is constructed by choosing a codeword for each segment from either \(A^0\) or \(A^1\). The codeword for the first segment is chosen from \(A^0\). The codeword for segment \(i = 2, \ldots, k\) is chosen as follows: if the last code bit in segment \((i-1)\) equals 0, then the codeword for segment \(i\) is chosen from \(A^1\); otherwise it is chosen from \(A^0\).

B. Rate

The rate of the above codes can be bounded from below as

\[
R \geq 1 - \frac{1}{b} \log_2(b + 1) - \frac{2}{b}.
\]

(49)

Indeed, there are \(2^{b-2}\) binary sequences of length \(b\) whose first two bits equal 0. Each of these sequences belongs to exactly one of the sets \(A^0_0, \ldots, A^0_b\). Therefore, the largest among these \((b + 1)\) sets will contain at least \(2^{b-2}/(b + 1)\) sequences and thus,

\[
|A^0_{a_0}| \geq \frac{2^{b-2}}{b + 1}.
\]

(50)

A similar argument gives the same lower bound for \(|A^1_{a_1}|\), hence

\[
M_s \geq \frac{2^{b-2}}{b + 1}.
\]

(51)

Taking logarithms gives (49).

From (49), we see that the rate penalty with respect to VT codes is at most \(\frac{2}{b}\) due to the prefix of length 2. As an example, for \(b = 16\) our code has 964 codewords, while the greedy
algorithm described in [1], gives 740; this is reduced to 652 when the search is restricted to VT codes. More examples are reported in Table I.

C. Decoding

Thanks to the segment-by-segment code construction, decoding will also proceed segment by segment. Decoding proceeds in the following simple steps.

In order to decode segment \( i \), for \( i = 1, \ldots, k \), assume that the first \( i - 1 \) segments have been decoded correctly. Thus the decoder knows the correct starting position of segment \( i \) in \( Y \); we denote it by \( p_i + 1 \).

By examining the last bit of segment \((i - 1)\), the decoder learns the correct syndrome for the codeword in segment \( i \), i.e., either \( a_0 \) or \( a_1 \); recall that segment 1 was drawn from \( \mathcal{A}^0 \). Without loss of generality, assume it is \( a_0 \); the decoding for \( a_1 \) is identical.

1) The decoder computes the VT syndrome

\[
\hat{a} = \text{syn}(Y(p_i + 1 : p_i + b))
\]

and compares it to the correct syndrome (assumed to be \( a_0 \)). There are two possibilities:

a) \( \hat{a} = a_0 \): The decoder concludes that there is no deletion in segment \( i \) and outputs \( \hat{S}_i = Y(p_i + 1 : p_i + b) \). The starting position of the next segment in \( Y \) is \( p_i + b + 1 \).

b) \( \hat{a} \neq a_0 \): The decoder knows there is a deletion in segment \( i \) and feeds \( Y(p_i + 1 : p_i + b - 1) \) to the VT decoder to recover the codeword. The output of the VT decoder is the decoded segment \( \hat{S}_i \). The starting position of the next segment in \( Y \) is \( p_i + b \).

2) The decoder now checks the last bit of the decoded segment \( \hat{s}_{i,b} \). If \( \hat{s}_{i,b} = 0 \), the decoder knows that segment \((i + 1)\) has been drawn from \( \mathcal{A}^1 \); otherwise it has been drawn from \( \mathcal{A}^0 \). Thus the decoder is now ready to decode segment \((i + 1)\).

D. Non-binary Code Construction

We now construct segmented deletion codes for alphabet size \( q > 2 \). For \( a = 0, \ldots, b - 1 \), and \( c = 0, \ldots, q - 1 \), define following sets:

\[
\mathcal{A}_{a,c}^j \triangleq \{ S \in \mathcal{X}^b : \text{syn}(A_S) = a, \sum(S) = c, s_1, s_2 \in \mathcal{X} \setminus \{ j \} \}.
\]

for \( j = 0, \ldots, q - 1 \). Now for each \( j = 0, \ldots, q - 1 \) define

\[
\{ a_j, c_j \} = \arg \max_{0 \leq a \leq b - 1, 0 \leq c \leq q - 1} |\mathcal{A}_{a,c}^j|.
\]
Similarly to the binary case, the sets \( A_{a_j,c_j}^j \) for \( 0 \leq j \leq q - 1 \) are used to construct the codebook. Choose the first segment from \( A_{a_0,c_0}^0 \). For encoding \( \ell \)th segment \((\ell > 1)\) we choose a word from \( A_{a_j,c_j}^j \) if \( j \) is the last symbol of segment \( \ell - 1 \). The size each set \( A_{a_j,c_j}^j \), for \( 0 \leq j \leq q - 1 \), can be bounded from below as \( M_s \geq \frac{q^{b-2}(q-1)^2}{qb} \).

Indeed, for any \( j \in \{0, (q - 1)\} \), there are \( q^{b-2}(q-1)^2 \) sequences of length \( b \) with the first two symbols are not equal to \( j \). Each of these symbols belong to one of the sets \( A_{a,c}^j \), where \( 0 \leq a \leq b - 1 \), and \( 0 \leq c \leq q - 1 \). Therefore the largest set has size at least \( \frac{q^{b-2}(q-1)^2}{qb} \). This gives a lower bound on the rate \( R \geq \log_2 \frac{q}{b} \log_2 b - \frac{1}{b} \log_2 q - \frac{2}{b} \log_2 \left( \frac{q}{q-1} \right) \).

Decoding proceeds in a similar way to the binary case. The main difference is that instead of computing \( 52 \), the decoder computes
\[
\hat{a} = \text{syn}(A_Z), \quad \hat{c} = \text{sum}(Z)
\]
where
\[
Z = Y(p_i + 1 : p_i + b).
\]

Then, the conditions in cases 1) a) and 1) b) are replaced by \( \{\hat{a} = a_0 \text{ and } \hat{c} = c_0\} \) and by \( \{\hat{a} \neq a_0 \text{ or } \hat{c} \neq c_0\} \), respectively.

V. SEGMENTED INSERTION CODES

A. Binary Code Construction

As in the deletion case, we define a subset of VT codewords such that upon decoding a segment, there is no ambiguity in the starting position of the next segment. We define the following set of sequences
\[
A_a \triangleq \{ S \in \{0, 1\}^b : \text{syn}(S) = a, s_1s_2 = 01, s_3s_4 \neq 01, S \neq 011 \cdots 1 \}
\]
and
\[
a_0 = \arg \max_{0 \leq a \leq b} |A_a|.
\]

Similarly to the previous section, the sets \( A_a \subseteq \mathcal{VT}(b) \) are sets of VT codewords with a prefix of a certain form. Our code is thus the maximal code in this family, i.e., \( C = A_{a_0}^k \). In contrast to the deletion case, the codeword for each segment is drawn from the same set \( A_{a_0} \).
In order to find the size of the code, we use similar arguments to those in the previous section. There are \(2^{b-2}\) sequences with prefix 01, out of which \(2^{b-4}\) are removed because they have prefix 0101; 01 \(\cdots\) 1 is excluded from \(A_a\) by construction. Each of the \(2^{b-2} - 2^{b-4} - 1\) sequences belong to exactly one of the sets \(A_0, \ldots, A_b\). Therefore, the largest of these \(b + 1\) sets will have size at least

\[
|A_{a_0}| \geq \frac{2^{b-2} - 2^{b-4} - 1}{b + 1}. \tag{61}
\]

This yields the following lower bound for the rate for \(b \geq 6\):

\[
R \geq 1 - \frac{1}{b} \log_2 (b + 1) - \frac{2.5}{b}. \tag{62}
\]

Hence the rate penalty is at most \(\frac{2.5}{b}\) due to the added constraints on the prefix.

B. Decoding

Decoding proceeds on a segment-by-segment basis, and as in the case of deletions, the code structure ensures that before decoding segment \(i\), the previous \((i - 1)\) segments have been correctly decoded. Thus the decoder knows the correct starting position of segment \(i\) in \(Y\); as before, denote it by \(p_i + 1\).

1) The decoder computes the VT syndrome

\[
\hat{a} = \text{syn}(Y(p_i + 1 : p_i + b)) \tag{63}
\]

and compares it to the correct syndrome \(a_0\). There are two possibilities:

a) \(\hat{a} \neq a_0\): The decoder knows that there has been an insertion in this segment and feeds \(Y(p_i + 1 : p_i + b + 1)\) to the VT decoder to recover the codeword. The output of the VT decoder is the decoded segment \(\hat{S}_i\). The decoder proceeds decoding segment \(i + 1\), skipping step 2. The starting position in \(Y\) for decoding segment \(i + 1\) is \(p_i + b + 2\).

b) \(\hat{a} = a_0\): The decoder concludes that there is no insertion in \(Y(p_i + 1 : p_i + b)\) and outputs \(\hat{S}_i = Y(p_i + 1 : p_i + b)\).

2) If case 1.b) holds, the decoder has to check whether \(y_{p_i+b+1}\) could be an inserted bit at the very end of the segment. To this end, the \(Y(p_i + b + 1 : p_i + b + 4)\) is checked against the prefix conditions for segment \(i + 1\) set in \(A_{a_0}\).

a) If \(y_{p_i+b+1} y_{p_i+b+2} \neq 01\): the decoder understands that there is an irregularity caused by either an insertion in \(y_{p_i+b+1}\), or in \(y_{p_i+b+2}\) or both. Therefore it deletes \(y_{p_i+b+1}\) and proceeds to decode segment \(i + 1\) starting from \(y_{p_i+b+2}\).
b) If \( y_{p_i+b+1}y_{p_i+b+2} = 01 \), \( y_{p_i+b+3}y_{p_i+b+4} \neq 01 \), then \( y_{p_i+b+1} \) is the correct start of segment \( i+1 \).

c) If \( y_{p_i+b+1}y_{p_i+b+2} = 01 \), \( y_{p_i+b+3}y_{p_i+b+4} = 01 \): In this case, the decoder needs to decide among three alternatives by decoding segment \( i+1 \):

i) \( y_{p_i+b+3} = 0 \) is an inserted bit in segment \( i+1 \) and no inserted bit in segment \( i \); let \( \tilde{Y}_1 = y_{p_i+b+1}y_{p_i+b+2}y_{p_i+b+4} \cdots y_{p_i+2b+1} \) denote the length \( b \) sequence resulting from deleting \( y_{p_i+b+3} \) from the received sequence. If \( \text{syn}(\tilde{Y}_1) = a_0 \) then \( \hat{S}_{i+1} = \tilde{Y}_1 \).

ii) \( y_{p_i+b+4} = 1 \) is an inserted bit in segment \( i+1 \) and no inserted bit in segment \( i \); let \( \tilde{Y}_2 = y_{p_i+b+1}y_{p_i+b+2}y_{p_i+b+3}y_{p_i+b+5} \cdots y_{p_i+2b+1} \) denote the length \( b \) sequence resulting from deleting \( y_{p_i+b+4} \) from the received sequence. If \( \text{syn}(\tilde{Y}_2) = a_0 \) then \( \hat{S}_{i+1} = \tilde{Y}_2 \).

iii) \( y_{p_i+b+1} = 0 \), \( y_{p_i+b+2} = 1 \) are inserted bits in segments \( i \) and \( i+1 \), respectively; let \( \tilde{Y}_3 = y_{p_i+b+3}y_{p_i+b+4} \cdots y_{p_i+2b+2} \) denote the length \( b \) sequence resulting from deleting \( y_{p_i+b+1}, y_{p_i+b+2} \) from the received sequence. If \( \text{syn}(\tilde{Y}_3) = a_0 \) then \( \hat{S}_{i+1} = \tilde{Y}_3 \).

When \( Y(b_i + 1 : b_i + 4) = 0101 \), we now show that the three cases listed in step 2.c) are mutually exclusive, and hence only one of them will give a matching VT syndrome. What needs to be checked is that the syndromes of \( \tilde{Y}_1, \tilde{Y}_2, \tilde{Y}_3 \) will all be different. From the very properties of VT codes we know that \( \text{syn}(\tilde{Y}_1) \neq \text{syn}(\tilde{Y}_2) \). Now find that

\[
\text{syn}(\tilde{Y}_1) - \text{syn}(\tilde{Y}_3) \pmod{(b+1)}
\]

\[
= \sum_{j=1}^{b} \tilde{y}_{1,j} - \sum_{j=1}^{b} \tilde{y}_{3,j} \pmod{(b+1)}
\]

\[
= 5 + \sum_{j=p_i+b+1}^{p_i+2b+1} y_j - 2 - by_{p_i+2b+2} \pmod{(b+1)}
\]

\[
= 3 + w_H(Y(p_i + b + 5 : p_i + 2b + 1)) + y_{p_i+2b+2} \pmod{(b+1)}
\]

\[
\neq 0
\]

where \( w_H(Z) \) denotes the Hamming weight of sequence \( Z \). The last step of (68) holds because

\[
3 + w_H(Y(p_i + b + 5 : p_i + 2b + 1)) + y_{p_i+2b+2} \pmod{(b+1)}
\]

can equal to 0 only if \( w_H(Y(p_i + b + 5 : p_i + 2b + 1)) = b - 3 \) and \( y_{p_i+2b+2} = 1 \), implying that both \( \tilde{Y}_1 = \tilde{Y}_3 = 011 \cdots 1 \). Since this sequence has been explicitly excluded from the codebook, we always have strict inequality, and hence \( \text{syn}(\tilde{Y}_1) \neq \text{syn}(\tilde{Y}_3) \). Furthermore, since

\[
\text{syn}(\tilde{Y}_2) - \text{syn}(\tilde{Y}_3) = \text{syn}(\tilde{Y}_1) - \text{syn}(\tilde{Y}_3) - 1
\]
is always non-zero, we conclude that there is no ambiguity at the decoder.

C. The Liu-Mitzenmacher conditions for binary segmented codes

In [1], Liu and Mitzenmacher specified three conditions such that any set of binary sequences satisfying these conditions is a zero-error code for both the segmented insertion channel and the segmented deletion channel. We list these conditions in Appendix B and show that the segmented insertion code $\mathcal{A}_{a_0}$ described in Sec. V-A satisfies these conditions. This shows that the segmented insertion code can also be used for the segmented deletion channel (though the decoder may be non-trivial).

It was conjectured in [1] that the rate and size of the maximal code satisfying the sufficient conditions grows with $b$. As our insertion code $\mathcal{A}_{a_0}$ satisfies the sufficient conditions, the lower bounds on its rate and size given in (61) and (62) confirm this conjecture.

D. Non-binary Code Construction

For the segmented insertion channel with alphabet size $q > 2$, we use prefix VT codes similar to those for the binary case. In this case, however, we set a prefix of length 3. This incurs a small penalty in rate with respect to the binary code described in Section V-A, but results in a slightly simpler decoder. Define the following sets for all $a = 0, \ldots, b - 1$ and $c = 0, \ldots, q - 1$

$$\mathcal{A}_{a,c} \triangleq \{ S \in \mathcal{X}^b : \text{syn}(A_S) = a, \text{sum}(S) = c, s_1s_2s_3 = 001 \}. \tag{71}$$

Now choose the largest set as the codebook, i.e., $\mathcal{C} = \mathcal{A}_{a_0,c_0}$ where

$$\{a_0, c_0\} = \arg \max_{0 \leq a \leq b-1, 0 \leq c \leq q-1} |\mathcal{A}_{a,c}|. \tag{72}$$

Similar to the binary case, the number of codewords can be bounded from below as

$$M_s \geq \frac{q^{b-3}}{q^b}, \tag{73}$$

which gives the following lower bound on the rate:

$$R \geq \log_2 q - \frac{1}{b} \log_2 b - \frac{4}{b} \log_2 q. \tag{74}$$

Decoding proceeds in a similar manner to the binary case. As the code is somewhat different from the binary one, we give a few more details about the decoder. Assume that the first $(i-1)$ segments have been decoded correctly, and let $p_i + 1$ is the starting point of the $i$th segment. Let $Z = Y(p_i + 1 : p_i + b)$, \tag{75}
and compute
\[
\hat{a} = \text{syn}(A_Z), \quad \hat{c} = \text{sum}(Z).
\] (76)

1) \(\hat{a} \neq a_0\) or \(\hat{c} \neq c_0\): The decoder knows there has been an insertion in the \(i\)th segment and feeds \(Y(p_i + 1: p_i + b + 1)\) to the non-binary VT decoder to recover the codeword. The output of the VT decoder is the decoded segment \(\hat{S}_i\). The starting position of the next segment in \(Y\) is \(p_i + b + 2\).

2) \(\hat{a} = a_0\) and \(\hat{c} = c_0\): The decoder concludes that there is no insertion in segment \(i\) and outputs \(\hat{S}_i = Y(p_i + 1: p_i + b)\). The decoder must then investigate the possibility of an insertion at the very end of the \(i\)th segment in order to find the correct starting point of the next segment. This is done as follows. First, if the symbol \(y_{p_i+b+1}\) is not equal to 0, it is an insertion. The decoder deletes the inserted symbol, and the starting position for the next segment is \((p_i + b + 2)\). Next, if \(y_{p_i+b+1} = 0\) and there is any symbol different from 0 or 1 in position \((p_i + b + 2)\) or \((p_i + b + 3)\), it is an inserted symbol thanks to the binary prefix. The decoder deletes the inserted symbol and sets the starting position of the next segment to \((p_i + b + 1)\). If neither of these cases hold, the decoder follows Table III.

| \(Y(p_i + b + 1: p_i + b + 3)\) | State of \(y_{p_i+b+1} = 0\) | Starting point of next segment |
|------------------|------------------|------------------|
| 001 | No action (\(y_{p_i+b+1}\) is not an insertion) | \(p_i + b + 1\) |
| 000 | Delete the first zero (\(y_{p_i+b+1}\) is an insertion) | \(p_i + b + 2\) |
| 010 | Delete the 1 (\(y_{p_i+b+1}\) may or may not be inserted) | \(p_i + b + 1\) |

VI. SEGMENTED INSERTION-DELETION CODES

A. Binary Code Construction

Since we now have both insertion and deletions, the decoder must first identify the type of edit in a segment prior to correcting it. Define the following sets:

\[
\mathcal{A}_a^0 \triangleq \{ S \in \{0,1\}^5 : \text{syn}(S) = a, s_1s_2s_3s_4s_5 = 00111, s_{b-2} = s_{b-1} = s_b \} \] (77)

\[
\mathcal{A}_a^1 \triangleq \{ S \in \{0,1\}^5 : \text{syn}(S) = a, s_1s_2s_3s_4s_5 = 11000, s_{b-2} = s_{b-1} = s_b \} . \] (78)
As in previous sections, these are subsets of VT codewords with certain constraints. In this case, in order to be able to identify the edit type, both prefix and suffix constraints have been added. Based on the above sets, we further define

$$a_0 = \arg \max_{0 \leq a \leq b} |A^0_a|, \quad a_1 = \arg \max_{0 \leq a \leq b} |A^1_a|$$

(79)

and \(M_s = \min \{|A^0_{a_0}|, |A^1_{a_1}|\}\). We construct the sets \(A^0, A^1\) by choosing \(M_s\) sequences from \(A^0_{a_0}, A^1_{a_1}\), respectively. Finally, the overall code of length \(n = kb\) is constructed by choosing a codeword for each segment from either \(A^0\) or \(A^1\). The codeword for the first segment is chosen from \(A^0\). For \(i \in \{2, \ldots, k\}\), if the last bit of segment \((i-1)\) is 0, then the codeword for segment \(i\) is drawn from \(A^1\) and otherwise from \(A^0\).

The size and rate are lower-bounded using the same arguments as in the previous sections. For \(b \geq 7\), we obtain

$$M_s \geq \frac{2^{b-7}}{b+1}$$

(80)

which yields a rate lower bound given by

$$R \geq 1 - \frac{1}{b} \log(b+1) - \frac{7}{b}.$$  

(81)

Due to the prefix and suffix constraints, our segmented insertion-deletion codes have a rate penalty of at most \(\frac{7}{b}\).

B. Decoding

As in the previous two cases, decoding proceeds segment-by-segment. We ensure that before decoding segment \(i\), the previous \((i-1)\) segments have all been correctly decoded. Hence, the decoder knows the correct starting position in \(Y\) for segment \(i\), which is denoted by \(p_i+1\). The decoder also knows whether \(S_i\) belongs to \(A^0\) or to \(A^1\). We discuss the case where \(S_i \in A^0\), so \(\text{syn}(S_i) = a_0\); the case where \(S_i \in A^1\) is similar, with the roles of the bits reversed.

The decoder computes the syndrome \(\text{syn}(Y(p_i+1 : p_i+b))\), and checks whether it equals \(a_0\). There are two possibilities:

1) \(\text{syn}(Y(p_i+1 : p_i+b)) \neq a_0\): This means that there is an edit in this segment, we should identify the type of edit and correct it. We show that can be done without ambiguity by using the fact that three last bits of each segment (suffix) are the same, and considering prefix of the next segment. The decoder’s decision for each combination of the three consecutive bits \((y_{p_i+b-1}, y_{p_i+b}, y_{p_i+b+1})\) is listed in Table III. Once the type of edit is known, the decoder
corrects the segment using the appropriate VT decoder. We now justify the decisions listed in Table III.

a) If \( y_{p_i+b-1} = y_{p_i+b} = y_{p_i+b+1} \): The edit is an insertion. To see this, assume by contradiction that it was a deletion. Then at least one of \( y_{p_i+b} \) and \( y_{p_i+b+1} \) are the first bit of the prefix of \( S_i+1 \), and \( y_{p_i+b-1} \) is a suffix bit of \( S_i \). This is not possible because by construction, the first two prefix bits of \( S_{i+1} \) must be different from the suffix bits of \( S_i \).

b) If \( y_{p_i+b-1} = y_{p_i+b} \neq y_{p_i+b+1} \): The edit is a deletion. To see this, suppose that the edit was an insertion; then the suffix condition can only be satisfied if \( y_{p_i+b+1} \) is the inserted bit. However, this implies that \( \text{syn}(Y(p_i+1:p_i+b)) \neq a_0 \), which is contradiction.

c) If \( y_{p_i+b-1} = y_{p_i+b+1} \neq y_{p_i+b} \): The edit could be either an insertion, or a deletion, according to the rules in lines 3, 4, 5 of Table III. If the edit is an insertion, then \( y_{p_i+b} \) is the inserted bit, therefore by omitting this bit, the sequence \( Z = [Y(p_i+1:p_i+b-1), y_{p_i+b+1}] \) should have VT-syndrome equal to \( a_0 \). Therefore, if \( \text{syn}(Z) \neq a_0 \), then the edit is deletion; if \( \text{syn}(Z) = a_0 \), we need to check the prefix of the next segment to determine the type of edit.

If \( \text{syn}(Z) = a_0 \): If \( y_{p_i+b+1} = y_{p_i+b+2} = y_{p_i+b+3} \), then the edit in segment \( i \) is a deletion (it can be verified that the prefix condition for segment \( (i+1) \) cannot otherwise be satisfied with at most one edit). In all other cases the edit in segment \( i \) is insertion, with \( y_{p_i+b} \) being the inserted bit. We observe that when \( \text{syn}(Z) = a_0 \), \( S_i = Z \) with either type of edit, but the decoder needs to infer the type of edit in order to guarantee the correct starting position for the next segment.

d) If \( y_{p_i+b-1} \neq y_{p_i+b} = y_{p_i+b+1} \): In this case, \( y_{p_i+b-2} \) determines the type of edit: if \( y_{p_i+b-2} =
TABLE IV: State of $y_{p_i+b+1}$ when $\text{syn}(Y(p_i+1:p_i+b)) = a_0$.

| $Y(p_i+b+1:p_i+b+5)$ | State of $y_{p_i+b+1}$ |
|------------------------|------------------------|
| 1uvst                  | Inserted               |
| 000uv                  | Inserted               |
| 011uv                  | Not Inserted           |
| 01000                  | Not possible           |
| 01001                  | Inserted               |
| 01010                  | Not possible           |
| 01011                  | Not inserted           |
| 00100                  | Not possible           |
| 00101                  | Not Inserted           |

00101 and $\text{syn}(Z_1)$ matches Inserted
00101 and $\text{syn}(Z_2)$ matches Not Inserted
00110                  | Not Inserted           |
00111                  | Not Inserted           |

$y_{p_i+b-1}$ the edit is a deletion, otherwise it is an insertion. This can be seen by examining the suffix condition: if the edit is an insertion then $y_{p_i+b-1}$ is the inserted bit therefore $y_{p_i+b-2}$ belongs to suffix of $S_i$, hence $y_{p_i+b-2} = y_{p_i+b} = y_{p_i+b+1}$. On the other hand, if the edit is a deletion, then $y_{p_i+b-2}$ and $y_{p_i+b-1}$ belongs to suffix of $S_i$, so they should be equal.

Hence we have shown that whenever $\text{syn}(Y(p_i+1:p_i+b)) \neq a_0$, we can uniquely decode $S_i$ and determine the correcting starting position for the next segment.

2) $\text{syn}(Y(p_i+1:p_i+b)) = a_0$: In this case $\hat{S}_i = Y(p_i+1:p_i+b)$. To determine the correct starting position for the next segment, we have to investigate the possibility of an insertion at the end of the block, i.e., determine whether $y_{p_i+b+1}$ is an inserted bit. This can be done by examining the prefix of $S_{i+1}$. We consider five bits, $Y(p_i+b+1:p_i+b+5)$, and for all 32 cases determine the state of $y_{p_i+b+1}$. For the simplicity, assume that the last bit of $S_i$ is 1, so that the prefix for $S_{i+1}$ is 00111; the other case is identical, with 0 and 1 interchanged.

First, if $y_{p_i+b+1} = 1$, then it is an inserted bit (this is 16 of the 32 cases). Table [V] lists the type of edit for each of the other cases when $y_{p_i+b+1} = 0$. These are justified below.

a) If $Y(p_i+b+1:p_i+b+5) = 011uv$ for some bits $u,v$, then $y_{p_i+b+1}$ is not an insertion corresponding to segment $i$: if it was inserted, then decoding for segment $(i+1)$ would start with the bits 11..., which cannot be matched with the prefix 00111 with only one
edit. Hence the correct starting position for decoding segment \((i + 1)\) is \(p_i + b + 1\).

b) If \(Y(p_i + b + 1 : p_i + b + 5) = 000uv\), then \(y_{p_i+b+1}\) (or another 0 from the run) is an insertion for segment \(i\), as 000u does not match 0011 unless we remove a zero from the run.

c) The cases \(Y(p_i + b + 1 : p_i + b + 5) = 01000, 01010, 00100\) cannot occur as they cannot be matched with the required prefix 00111 through any valid edits for segment \(i + 1\), whether or not \(y_{p_i+b+1}\) is inserted.

d) If \(Y(p_i + b + 1 : p_i + b + 5) = 01001\), then \(y_{p_i+b+1}\) is an insertion for segment \(i\) as this is the only option consistent with the prefix 00111.

e) If \(Y(p_i + b + 1 : p_i + b + 5) = 0011u\) or 01011, then \(y_{p_i+b+1} = 0\) is not an insertion for segment \(i\), and is the starting bit for decoding segment \((i + 1)\).

f) If \(Y(p_i + b + 1 : p_i + b + 5) = 00101\), we need to compare the VT syndromes of two sequences to determine the status of \(y_{p_i+b+1}\). We will also decode \(S_{i+1}\) in the process. If \(y_{p_i+b+1} = 0\) is inserted, then \(y_{p_i+b+3} = 1\) should also be inserted, therefore \(S_{i+1} = Z_1\) where \(Z_1 = [00, Y(p_i + b + 5 : p_i + 2b + 2)]\). On the other hand, if \(y_{p_i+b+1}\) is not inserted then \(y_{p_i+b+4} = 0\) should be an inserted bit, therefore, \(S_{i+1} = Z_2\) where \(Z_2 = [001, Y(p_i + b + 5 : p_i + 2b + 1)]\). However, \(Z_1\) and \(Z_2\) will always produce different syndromes and only one of them will be equal to \(a_0\), the correct syndrome for segment \((i + 1)\). Thus we can correctly identify whether \(y_{p_i+b+1}\) was an insertion for segment \(i\) or not.

Hence we have shown that whenever \(\text{syn}(Y(p_i + 1 : p_i + b)) = a_0\), we can uniquely decode \(S_i\) and determine the correcting starting position for the next segment.

C. Non-binary Code Construction

We now construct segmented insertion-deletion codes for alphabet size \(q > 2\). For \(a = 0, \ldots, b - 1\), and \(c = 0, \ldots, q - 1\), define following sets:

\[
\mathcal{A}_{a,c}^0 \triangleq \{ S \in \mathcal{X}^b : \text{syn}(A_S) = a, \text{sum}(S) = c, s_1s_2s_3s_4s_5 = 00111, s_{b-2} = s_{b-1} = s_b \},
\]

(82)

\[
\mathcal{A}_{a,c}^1 \triangleq \{ S \in \mathcal{X}^b : \text{syn}(A_S) = a, \text{sum}(S) = c, s_1s_2s_3s_4s_5 = 11000, s_{b-2} = s_{b-1} = s_b \}.
\]

(83)

For \(j = 0, 1\) define

\[
\{ a_j, c_j \} = \arg \max_{0 \leq a \leq b - 1, 0 \leq c \leq q - 1} |\mathcal{A}_{a,c}^j|,
\]

(84)
We use the sets $A^0_{a_0,c_0}$ and $A^1_{a_1,c_1}$ to construct the codebook by alternating depending on the last symbol of the previous segment. We set $M_s = \min\{A^0_{a_0,c_0}, A^1_{a_1,c_1}\}$ and construct the sets $A^0, A^1$ by choosing $M_s$ sequences from $A^0_{a_0,c_0}, A^1_{a_1,c_1}$, respectively. The codeword for the first segment is chosen from $A^0$. For $i \in \{2, \ldots, k\}$, if the last symbol of segment $(i - 1)$ is an even number, then the codeword for segment $i$ is drawn from $A^1$; if the last symbol of segment $(i - 1)$ is an odd number, the codeword is drawn from $A^0$.

The number of codewords per segment satisfies

$$M_s \geq \frac{q^{b-7}}{qb}$$

and thus a lower bound on the rate is

$$R \geq \log_2 q - \frac{1}{b} \log_2 b - \frac{8}{b} \log_2 q.$$  (86)

The decoding is almost identical to the binary case. As with previous decoders, to decode segment $i$, it is assumed that the first $(i - 1)$ segments have been decoded correctly. Let $Z = Y(p_i + 1 : p_i + b)$, where $p_i + 1$ is the starting position of the $i$th segment. Compute

$$\hat{a} = \text{syn}(A_Z), \quad \hat{c} = \text{sum}(Z).$$

(87)

The decoder checks whether $\{\hat{a} = a_0 \text{ and } \hat{c} = c_0\}$ or $\{\hat{a} \neq a_0 \text{ or } \hat{c} \neq c_0\}$. In the first case, the decoder sets $\hat{S}_i = Y(p_i + 1 : p_i + b)$ and in order to find the starting point of segment $i + 1$, follows the same case breakdown as in the binary decoder (see case 2 of the binary decoder). On the other hand, if $\{\hat{a} \neq a_0 \text{ or } \hat{c} \neq c_0\}$, thanks to the prefix-suffix code structure being the same as the binary one, the decoder follows exactly the same case breakdown (see case 1 of the binary decoder) in order to identify the type of edit and correct it.

APPENDIX

A. Proof of (31)

Let $U$ be a Binomial $\left(b, \frac{q-1}{q}\right)$ random variable, with mean $\mu = \frac{b(q-1)}{q}$. Then, using a standard Chernoff bound for a binomial random variable (see, for example [14 Theorem 4.5]), we have for any $\epsilon > 0$:

$$\mathbb{P}(U \leq \mu(1 - \epsilon)) \leq \exp\left(\frac{-\mu\epsilon^2}{2}\right).$$

(88)
Choosing $\epsilon = \sqrt{\frac{2}\kappa q \ln b/(q-1)b}$, we have

$$
\mu(1-\epsilon) = \frac{b(q-1)}{q} - \sqrt{\frac{2\kappa(q-1)b\ln b}{q}}
$$

(89)

$$
= r,
$$

(90)

where $r$ is defined in (25). Using this in (88), we obtain

$$
P(U \leq r) = P(U \leq \mu(1-\epsilon))
$$

(91)

$$
\leq \exp \left( -\frac{\mu \epsilon^2}{2} \right)
$$

(92)

$$
= b^{-\kappa},
$$

(93)

where the last equality is obtained by substituting the values of $\mu$ and $\epsilon$. Finally, note that

$$
P(U \leq r) \geq P(U \leq (r-1))
$$

(94)

$$
\geq \sum_{t=0}^{r-1} \left( 1 - \frac{1}{q} \right)^t \left( \frac{1}{q} \right)^{b-t} \binom{b}{t}
$$

Combining (94) and (93) yields the desired inequality.

B. The Liu-Mitzenmacher conditions

Let $\mathcal{I}_1(X)$ denote the set of all sequences obtained by adding one bit to the binary sequence $X$. Then $C \subseteq \{0,1\}^b$ is a binary zero-error code for both the segmented insertion channel and the segmented deletion channel (with segment length $b$) if the following conditions are satisfied.

1) For any $U, V \in C$, with $U \neq V$, $\mathcal{I}_1(U) \cap \mathcal{I}_1(V) = \emptyset$;

2) For any $U, V \in C$, with $U \neq V$, prefix($\mathcal{I}_1(U)$) $\cap$ suffix($\mathcal{I}_1(V)$) = $\emptyset$;

3) Any string of the form $y^*(zy)^*$ or $y^*(zy)^*z$, where $y, z \in \{0,1\}$, is not in $C$.

Here prefix($X$) denotes the subsequence of $X$ obtained excluding the last bit, suffix($X$) the subsequence obtained excluding the first bit, and $X^*$ is the regular expression notation referring to 0 or more copies of sequence $X$. The set prefix($\mathcal{I}_1(U)$) is defined as $\{\text{prefix}(X): X \in \mathcal{I}_1(U)\}$. The set suffix($\mathcal{I}_1(V)$) is defined similarly.

We now show that the insertion code $\mathcal{A}_{a_0}$ defined in Sec. V-A satisfies these conditions. Since $\mathcal{A}_{a_0}$ is a subset of a VT code and is hence a single insertion correcting code, the first condition is satisfied.
We next verify the third condition. All the codewords in $A_{a_0}$ start with 01. It is easy to see that any sequence starting with 01 and violating the third condition in either of the two ways must have 0101 as its first four bits. But these sequences are excluded from $A_{a_0}$, so each codeword in $A_{a_0}$ satisfies the third condition.

It remains to prove that the second condition is satisfied. Assume towards contradiction that there exist codewords $U, V \in A_{a_0}$ such that $U \neq V$ and the set $W = \text{prefix}(I_1(U)) \cap \text{suffix}(I_1(V))$ is non-empty. Suppose that the sequence $Z \in W$, and $Z_1 \in I_1(U)$ and $Z_2 \in I_1(V)$ are length $(b + 1)$ sequences such that that $Z = \text{prefix}(Z_1) = \text{suffix}(Z_2)$.

Since $U \in A_{a_0}$ and $Z_1 \in I_1(U)$, $\text{prefix}(Z_1)$ will start with a 0, unless the inserted bit in $Z_1$ is a 1 and is inserted exactly at the beginning of $U$, i.e., unless $Z_1 = [1, U]$. Also, since $Z_2 \in I_1(V)$, $\text{suffix}(Z_2)$ will start with 1 unless $Z_2$ is obtained by adding a bit at the beginning of $V$, i.e. $Z_2 = [h, V]$, for $h \in \{0, 1\}$. Since $Z = \text{prefix}(Z_1) = \text{suffix}(Z_2)$, clearly one of the above two cases should hold. First, assume that $Z$ starts with 1 and therefore we have $Z_1 = [1, U]$. Now since $U \in A_{a_0}$ starts with 01, we have

\[
Z = \text{prefix}(Z_1) = Z_1(1 : b) = [1, U(1 : b - 1)] = [101, U(3 : b - 1)].
\]

Now we also know that $Z = \text{suffix}(Z_2)$, so $\text{suffix}(Z_2) = [101, U(3 : b - 1)]$. Now, notice that $Z_2 \in I_1(V)$ and first bit of $V$ is 0, so the first two bits of $Z_2$ cannot be 11. We therefore have

\[
Z_2 = [0101, U(3 : b - 1)].
\]

But we know that $V \in A_{a_0}$ cannot start with 0101, so either the third or the fourth bit in $Z_2$ is the inserted bit. Therefore, we know that

\[
V = [01z, U(3 : b - 1)],
\]

for $z \in \{0, 1\}$. We also know that

\[
U = [01, U(3 : b - 1), u_b],
\]

where $u_b \in \{0, 1\}$. But this contradicts condition 1 (which has already been verified) because we obtain the same length $(b + 1)$ sequence by: i) inserting $u_b$ to the end of $V$, and ii) inserting $z$ after the second bit of $U$. 

Next consider the second case where $Z$ starts with a 0. As explained above, we then have $Z_2 = [h, V]$, and hence, $Z = \text{suffix}(Z_2) = V$. Therefore $\text{prefix}(Z_1) = V$, so one can obtain $Z_1$ by adding the last bit of $Z_1$ to $V$. Therefore $Z_1 \in I_1(U) \cap I_1(V)$, which is a contradiction. This completes the proof that $A_{a_0}$ satisfies all the three conditions.

**REFERENCES**

[1] Z. Liu and M. Mitzenmacher, “Codes for deletion and insertion channels with segmented errors,” *IEEE Trans Inf. Theory*, vol. 56, no. 1, pp. 224–232, 2010.

[2] N. J. A. Sloane, “On single-deletion-correcting codes,” in *Codes and Designs, Ohio State University (Ray-Chaudhuri Festschrift)*, pp. 273–291, 2000. Online: [https://arxiv.org/abs/math/0207197](https://arxiv.org/abs/math/0207197).

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

[4] E. A. Ratzer, “Marker codes for channels with insertions and deletions,” *Ann. Telecommun.*, vol. 60, no. 1, pp. 29–44, 2005.

[5] K. A. Abdel-Ghaffar, F. Pulunic, H. C. Ferreira, and W. A. Clarke, “On Helberg’s generalization of the Levenshtein code for multiple deletion/insertion error correction,” *IEEE Trans on Inf. Theory*, vol. 58, no. 3, pp. 1804–1808, 2012.

[6] D. Cullina, N. Kiyavash, and A. A. Kulkarni, “Restricted composition deletion correcting codes,” *IEEE Trans Inf. Theory*, vol. 62, no. 9, pp. 4819–4832, 2016.

[7] J. Brakensiek, V. Guruswami, and S. Zbarsky, “Efficient low-redundancy codes for correcting multiple deletions,” in *Proc. Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1884–1892, 2016.

[8] V. Guruswami and C. Wang, “Deletion codes in the high-noise and high-rate regimes,” *IEEE Trans Inf. Theory*, vol. 63, no. 4, pp. 1961–1970, 2017.

[9] F. Sala, R. Gabrys, C. Schoeny, and L. Dolecek, “Exact reconstruction from insertions in synchronization codes,” *IEEE Trans Inf. Theory*, vol. 63, no. 4, pp. 2428–2445, 2017.

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

[11] G. Tenengolts, “Nonbinary codes, correcting single deletion or insertion,” *IEEE Trans on Inf. Theory*, vol. 30, no. 5, pp. 766–769, 1984.

[12] A. Kulkarni, “Insertion and deletion errors with a forbidden symbol,” in *Proc. IEEE Inf. Theory Workshop*, 2014.

[13] M. Abroshan, R. Venkataramanan, and A. Guillén i Fàbregas, “Efficient systematic encoding of VT codes,” 2017. (Online) [http://arxiv.org/abs/1708.04071](http://arxiv.org/abs/1708.04071).

[14] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized algorithms and probabilistic analysis*. Cambridge Univ. Press, 2005.