Insdel Codes with Synchronization Strings: Error-Correction in Insertion-Deletion Channels

Nicholas Sacco
Research Advisor: Dr. Bobak Nazer
Committee: Dr. Ashok Cutkosky, Dr. Venkatesh Saligrama, Dr. Ari Trachtenberg

Abstract

Codes designed to correct for erasures and corruptions are not effective at correcting for insertions and deletions. Haeupler and Shahrasbi propose the "insdel code," a scheme that augments ordinary error-correcting codes with an indexing sequence, enabling data recovery in the presence of a fixed number of insertions and deletions. Their proposed indexing sequence, the ε-synchronization string, enables the creation of codes with finite-sized alphabets, constant rates for large n, and polynomial-time construction and decoding algorithms. Communication over an insertion-deletion channel was simulated; synchronization strings demonstrated several advantages compared to unique indexing sequences, indicating they are suitable for practical applications.

Index Terms

Synchronization strings, unique indexing, insertion-deletion channel, synchronization errors

Full paper: B. Haeupler and A. Shahrasbi, "Synchronization strings: codes for insertions and deletions approaching the Singleton bound," in Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing. Montreal Canada: ACM, Jun. 2017, pp. 33–46. [Online]. Available: https://dl.acm.org/doi/10.1145/3055399.3055498

I. Introduction

One of the most widely studied and adopted mechanisms for recovering data over a noisy channel is the error-correcting code. Much of the historical research has focused on the development of codes that correct for symbol erasures and corruptions (Fig. 1), [1]. However, error regimes exist where these types of codes are ineffective. Haeupler and Shahrasbi consider communication of the string S of length n over an (n,δ)-insertion-deletion channel, where the received string S' is corrupted by at most $n\delta$ symbol insertions and deletions (Fig. 2). A code must be developed that corrects for these "synchronization" errors [1].

Development of codes to correct for insertions and deletions started roughly in parallel with codes for erasures and corruptions, with the first major theoretical contribution by Levenshtein [2] in 1965, who demonstrated that codes correcting for insertions and deletions are possible. Progress continued, mostly focused on narrow theoretical cases, such as a single insertion/deletion [3], and practical applications, such as codes focusing on insertions and deletions in RLL-coding applications, like magnetic storage media [4], [5], [6]. Additional work generalized schemes to consider regimes of multiple insertions, deletions, and even corruptions [7], but the main leap forward was the first "asymptotically good" codes proposed by Schulman and Zuckerman in 1999 [4]. They proposed an un-optimized, buffered and indexed, concatenated coding scheme that enabled correction of insertions and deletions, and more importantly, demonstrated constant rate and distance [1], [4]. Improvements to the bounds on the rate in a variety of noise regimes were made by Guruswami and Liu in 2016, building from the same style of buffered, indexed, concatenated coding schemes [8]. Haeupler and Shahrasbi continued the improvement trend; this paper will discuss their intermediate results, the synchronization string, and its application to the construction of "insdel codes."

II. Proposed Protocol: Insdel Codes

The construction procedure for the authors' "insdel codes," correcting for $n\delta$ insertions and deletions, is outlined below (Fig. 3), [1]:

- Encode data d using the code $\mathcal{C}$, which "efficiently corrects" for erasures and corruptions: $\mathcal{E}_{\mathcal{C}}(d) = c$.
- Generate an "indexing sequence" S with symbols drawn from the indexing alphabet $\Sigma_{\mathcal{S}}$. For each codeword element c_t, attach the corresponding indexing element $S[t]$, producing tuples $[(c_t, S[t])]$.
- Transmit the n tuples over the channel; n' tuples $[(c'_{t'}, S'[t'])]$ are received, where $|n-n'| \leq n\delta$.
- The received indexing sequence S' is a corrupted version of the original sequence S. Using the indexing decoder $\mathcal{D}_{\mathcal{S}}$, map the $n\delta$ insertions and deletions that appear in S' to h symbol erasures appearing in the codeword sequence: $\mathcal{D}_{\mathcal{S}}(S', c') = \tilde{c}$.
- The receiver has a version of the original codeword c as if it were corrupted **solely** by symbol erasures. Provided the number of induced erasures is small enough, the base decoder can recover the original message from the corrupted codeword: $\mathcal{D}_{\mathcal{C}}(\tilde{c}) = d$.

The primary innovation in this scheme is the decoder $\mathcal{D}_{\mathcal{S}}$, which maps insertions and deletions in the received indexing sequence S' to erasures in the corrupted codeword $\tilde{c}$. Synchronization errors may be hard to correct, but if there exists a procedure that "converts" $n\delta$ insertions and deletions into $h = f(n,\delta)$ symbol erasures, recovering the original message is as simple as selecting an appropriate base code $\mathcal{C}$ with distance $d_{\mathcal{C}} \geq h+1$. The main objective is to create an efficient indexing sequence $S \in \Sigma_{\mathcal{S}}^n$, decoding algorithm $\mathcal{D}_{\mathcal{S}}$, and mapping $h = f(n,\delta)$.

III. Potential Indexing Schemes

A naive first attempt is the "unique indexing scheme," where the indexing sequence S consists of n unique elements. The decoding algorithm is simple: map missing indices (deletions) and duplicate indices (insertions) to symbol erasures. At most $h = f(n,\delta) \approx n\delta$ erasures are induced in the corrupted codeword $\tilde{c}$; any base code C with rate R_C that can correct for this number of erasures is sufficient. Construction and decoding algorithms can be naively implemented in $T_{S_{ENC}} = O(n)$ and $T_{S_{DEC}} = O(n^2)$ time (Figs. 5, 6).

While this simple method yields efficient construction and decoding times, the indexing overhead associated with this method is sub-optimal for large n. Since each unique index $S[t]$ is represented with $\log n$ bits, the overall rate of the insdel code is reduced by a factor of $\log n$: $R_{\mathcal{S}} = \frac{R_{\mathcal{C}}}{\log n}$. A new indexing scheme must be developed to avoid this asymptotic rate reduction.

The proposed ε-synchronization string, $\varepsilon \in (0,1)$, is an indexing sequence that avoids this problem. S is a ε-synchronization string if all substrings $S[i,k)$ satisfy the edit distance requirement: $ED\{S[i,j), S[j,k)\} > (1-\varepsilon)(k-i)$, $i < j < k$. While this condition is much stricter than the unique indexing scheme, valid ε-synchronization strings of length n can be constructed from an alphabet of size $|\Sigma_{\mathcal{S}}| = \Theta(\varepsilon^{-4})$. Notably, the alphabet $\Sigma_{\mathcal{S}}$ is both finite-sized and independent of the transmission length n, both desirable properties.

Like the unique indexing scheme, essentially any base code $\mathcal{C}$ with rate $R_{\mathcal{C}}$ can be used, provided the base code corrects for $h = f(n,\delta,\varepsilon) \leq \left(\frac{5-\varepsilon}{1-\varepsilon}\right) n\delta$ erasures. The rate of the insdel code using synchronization strings is preserved much more efficiently due to the alphabet's independence of n, with $R_{\mathcal{S}} = R_{\mathcal{C}}\left(1 - \frac{\log|\Sigma_{\mathcal{S}}|}{\log|\Sigma_{\mathcal{C}}|}\right)$.

The construction algorithm starts by generating a random string of length n drawn from Σ_S. Each interval is verified to ensure the ε-synchronization property holds. Any interval that does not satisfy the property is replaced with a new, randomly generated interval. The procedure repeats until no violating intervals are found. A compliant ε-synchronization string is produced in polynomial time, with $T_{S_{ENC}} = O(n^5)$ (Fig. 7).

The decoding algorithm utilizes the authors' proposed Relative Suffix Distance (RSD), a weighted edit distance metric. The decoder attempts to match each received substring $S'[1,j]$ to a prefix $S[1,i]$ of the expected indexing sequence. For each substring $S'[1,j]$, the RSD is computed between itself and all prefixes of S. The index i that minimizes the value of $RSD\{S'[1,j],S[1,k]\}$ over all k is returned, with the interpretation that the received substring $S'[1,j]$ was **most likely** transmitted as the original prefix $S[1,i]$. Like the unique indexing scheme, all missing or duplicate indices in the decoded list are mapped to erasures in the reconstructed codeword $\tilde{c}$, completing the indexing decoding phase. This "Minimum RSD Decoding" is completed in polynomial time, with $T_{S_{DEC}} = O(n^4)$ (Fig. 8).

IV. Case Study and Discussion

Several simulations were conducted to compare the practical deployment of both methods. A Reed-Solomon code was used as the base code $\mathcal{C}$ [9]. The main results from the Case Study are summarized below.

Efficiency: Synchronization strings are more efficient than unique indexing schemes if, for the same insdel code rate R_S, fewer bits are required to transmit the data and index elements over the channel. For the code $\mathcal{C}$, with $n > 113$, and carefully selected $\varepsilon > \varepsilon_R$, the equivalent rate ε, using a synchronization string is more efficient than a unique indexing sequence (Fig. 9).

Probability of Error: Four tests were carried out, using both indexing schemes and two error models: 'Fixed," where exactly $n\delta$ insertions/deletions are applied, and "I.I.D.," where the probability of applying an insertion/deletion was δ. The channel noise δ was varied, and at each noise level, the probability of error p_{error} in recovering the original transmitted message was computed. The same code $\mathcal{C}$ was used with R_C and ε selected to fix rate R_S in all tests. In both error models, the synchronization string broadly achieved a lower p_{error} over a wider range of δ, demonstrating the synchronization string is more resilient to channel noise and more robust in various error models than the unique indexing sequence (Figs. 10, 11, 12, 13).

Performance: Synchronization strings of various lengths were constructed and decoded; ε was selected such that the rate R_S was fixed regardless of the scheme. As expected, large synchronization strings required construction and decoding times impractical for real-time applications, unlike the more efficient unique indexing method. The large construction time is likely due to the probabilistic construction algorithm. With low probability of generating valid strings for large n, an extensive search is required (Figs. 14, 15).

Insdel codes show theoretical gains over prior schemes, such as improvements to the rate-distance trade-off and elegant construction/decoding algorithms. Further innovation on the structure yielded additional gains: using ε-self matching strings, the rate-distance trade-off was improved, and construction/decoding times on par with unique indexing were achieved [1]. Using ε-synchronization circles, the size of the alphabet was reduced to $|\Sigma_S| = O(\varepsilon^{-2})$ [10]. Even without improvements, insdel codes based on ε-synchronization strings exhibit a more efficient rate R_S, alphabet size $|\Sigma_S|$, and noise resilience than the unique indexing scheme, at the cost of longer construction and decoding. If these times can be mitigated, practical codes rivalling unique indexing implementations are possible.

V. Future Work and Applications

Given the advantages of synchronization strings and their use over general insertion-deletion channels, a valid research direction might focus on applying synchronization strings, or some similarly designed structure, in problems modeled as a variation of the insertion-deletion channel. Leahy et. al. were inspired by ε-synchronization strings, and directly adopted the structure for quantum insertion-deletion channels [11]. Shomorony and Heckel studied DNA-based storage, and originally proposed using unique indexing to correct for errors over a proposed "shuffling channel" model [12]. Anker et. al. studied the use of codes to correct for dropped packets in a TCP communication, attempting to minimize retransmissions [13]. This network can be modeled as a narrower deletion channel. In these cases, synchronization strings are strong candidates for necessary indexing sequences, and might readily be adopted for these vastly different computing regimes. And if not, then ε-synchronization strings might serve as the inspiration for new and efficient indexing sequences for specific channel models and error regimes.

Practical deployment of synchronization strings might be made feasible using a "synchronization string API" hosted on a computing cluster. At run-time, users request precomputed strings of the required length and ε. Received data is submitted for decoding, which utilizes a parallelized version of the minimum RSD algorithm running on the cluster's resources. While real-time performance may not be guaranteed, major improvements to construction and decoding times can be made to enable practical use of insdel codes.

VI. Appendix

A. Channel Models

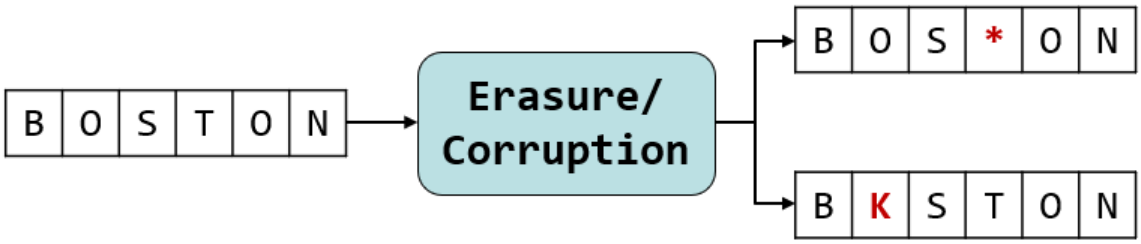

Fig. 1: *A channel susceptible to symbol erasures and symbol corruptions. In the transmitted word BOSTON, the symbol T is erased (erasure), while the first symbol O is corrupted and replaced with the symbol K (corruption).*

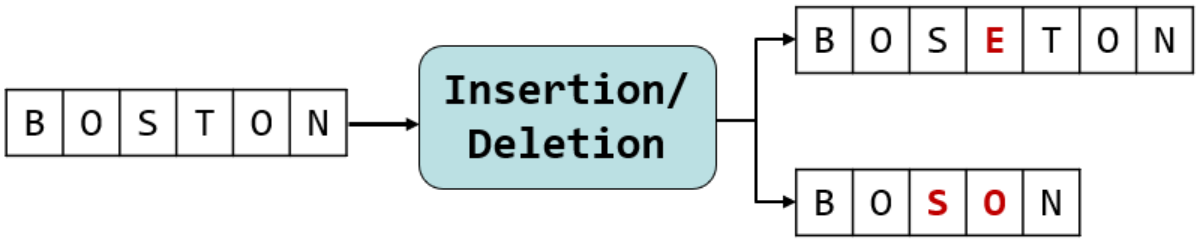

Fig. 2: *A channel susceptible to symbol insertions and deletions. In the transmitted word BOSTON, the symbol E is inserted into the stream (insertion), while the symbol T is deleted from the stream (deletion).*

B. Insdel Codes: Construction and Decoding

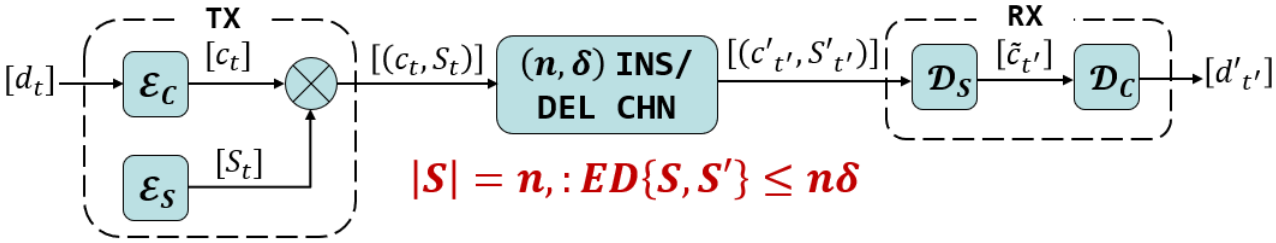

Fig. 3: *Block diagram representation of the proposed insdel code. Data d is encoded using the base encoder $\mathcal{E}_C$, producing the codeword c. Indexing sequence S is generated from the indexing encoder $\mathcal{E}_S$; each element S_t is attached to the corresponding element in the codeword, producing a list of tuples $[(c_t, S_t)]$. The n tuples are transmitted over the (n, δ)-Insertion-Deletion Channel, over which at most $n\delta$ insertions or deletions are introduced into the transmission of length n. At the receiver, n' tuples are received of the form $[(c'_{t'}, S'_{t'})]$. Formally, the (n, δ)-Insertion-Deletion channel guarantees that the edit distance between the transmitted indexing sequence S and the received indexing sequence S' is bounded by $n\delta$: $ED\{S, S'\} \leq n\delta$. The received tuples are then decoded using the indexing decoder $\mathcal{D}_S$, which maps insertions and deletions that occur in the received indexing sequence S' into symbol erasures appearing the codeword sequence, yielding the corrupted codeword $\tilde{c}$. With insertions and deletions mapped back to erasures, the base decoder $\mathcal{D}_C$ can decode the corrupted codeword, presuming the number of erasures is within the error-correcting capabilities of the code C. [1], [14].*

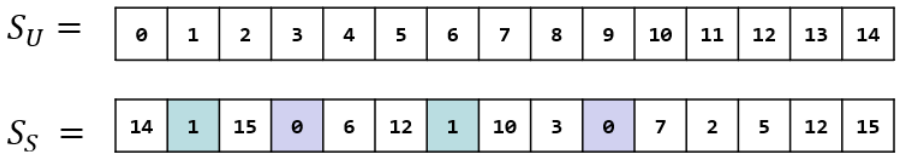

Fig. 4: *Examples of indexing sequences S in both the unique indexing method (top) and the synchronization string method (bottom). Each sequence is of length $n = 15$, and the synchronization string was created with $\varepsilon = 0.5$; the required alphabet size for such a value is $|\Sigma_S| = \varepsilon^{-4} = 16$ elements. Note that several elements are repeated or missing in the synchronization string; for example, elements 0 and 1 appear twice, and element 4 is not found in the synchronization string. Reusing elements in the indexing sequence S enables the synchronization string to be created from an alphabet with size independent of the stream length n.*

Algorithm: Unique Indexing Construction $\mathcal{E}_S$

```
codeword = [c_t]
augmented_codeword = [ ]

for each c_t in codeword:
      augmented_codeword[t] = (c_t, t)
return augmented_codeword
```

Fig. 5: ***Unique Indexing Construction Algorithm*** *$\mathcal{E}_S$. Essentially, for each element c_t in the codeword, attach the index t. If the codeword is length n, then this algorithm runs in $O(n)$ time, requiring a single pass through the codeword to attach a unique indexing symbol [1], [14].*

Algorithm: Unique Indexing Decoder $\mathcal{D}_S$

```
expected_indices = {1, 2, ..., n}
received_tuples = [(c'_t', S'_t')]
reconstructed_codeword = { }

For each index i in expected_indices:
     If i not in [S'_t']:
          DELETION; reconstructed_codeword[i] = ERASURE;
     If i in [S'_t'] and count(i) > 1:
          INSERTION; reconstructed_codeword[i] = ERASURE;
     If i in [S'_t'] and count(i) == 1:
          No apparent error; reconstructed_codeword[i] = c'_t'
return reconstructed_codeword
```

Fig. 6: ***Unique Indexing Decoding Algorithm*** *$\mathcal{D}_S$. If there were no synchronization errors, then each unique index would appear exactly once in the received indexing sequence S'. Therefore, the decoding algorithm can be designed as follows: for each expected index t in S, check to see if t appears in the received sequence S' exactly once. If it is not found exactly once (0 times for a deletion, > 1 times for an insertion), then add an erasure symbol in the reconstructed codeword: $\tilde{c}_t = *$. Otherwise, add in the corresponding symbol: $\tilde{c}_t = c'[loc(t)]$, where $loc(t)$ returns the location of index t within the received indexing sequence S'. This approach naively requires n passes through the received sequence S', one for each index in S. We still expect $S' = O(n)$, so the overall run time of this naive algorithm is $O(n^2)$ [1], [14].*

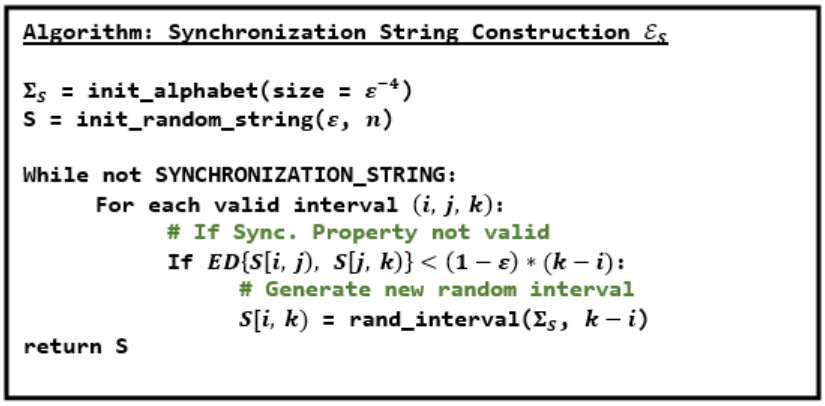

Fig. 7: ε-***Synchronization String Construction Algorithm*** $\mathcal{E}_S$. *The algorithm starts by initializing a random string of length n, with elements drawn from the indexing alphabet* Σ_S, *where* $|\Sigma_S| = \Theta(\varepsilon^{-4})$ *elements. While the string is not compliant, the* ε*-synchronization property is checked for each interval in the string (Table I). If the property is not satisfied, then a new random interval is generated. The verification repeats until all intervals satisfy the* ε*-synchronization property. This algorithm is inherently probabilistic; the authors state that* $O(n)$ *iterations are expected to generate a valid string [1]. There are* $O(n^2)$ *intervals in the string, each requiring an edit distance computation with* $O(n^2)$ *run time. Therefore, the overall run time of this probabilistic algorithm is* $O(n^5)$ *[1], [14].*

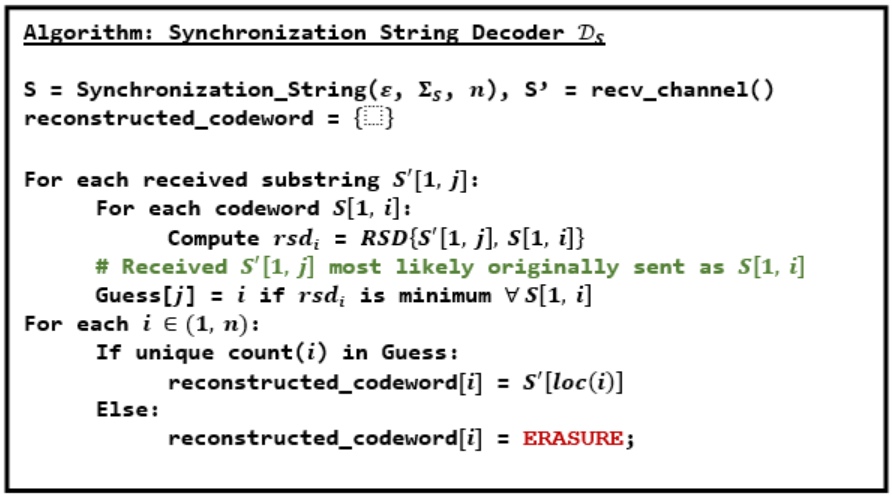

Fig. 8: ε-***Synchronization String Decoding Algorithm*** $\mathcal{D}_S$. *The receiver possesses both the expected indexing sequence S, and the received indexing sequence* S'. *Using both sequences in the Minimum Relative Suffix Distance Decoding (MRSDD) Algorithm, the receiver can decode from synchronization errors. For each substring* $S'[1, j]$ *in the received indexing sequence* S', *the RSD is computed to all prefixes* $S[1,k]$ *of the expected indexing sequence S:* $rsd_k = RSD\{S'[1, j], S[1,k]\}\ \forall k$. *The MRSDD algorithm returns index i if* $rsd_i < RSD\{S'[1, j], S[1,k]\}\ \forall k \neq i$ *- in other words, the relative suffix distance between the received substring* $S'[1, j]$ *was minimized with the expected substring* $S[1,i]$. *The interpretation is as follows: The decoder estimates that the received substring* $S'[1, j]$ *was most likely originally transmitted as the substring* $S[1,i]$, *but due to insertions and deletions, was corrupted and received as* $S'[1, j]$. *With the list of decoded indices, the mapping of synchronization errors to erasures exactly follows the same procedure as the unique indexing scheme. Indices that do not appear, or appear multiple times, are mapped to erasures. Otherwise, the codeword is built up from valid symbols:* $\tilde{c}_t = c'[loc(t)]$. *There are* $O(n)$ *substrings* $S'[1, j]$ *and* $O(n)$ *substrings* $S[1,i]$, *so the number of required RSD computations is* $O(n^2)$; *each computation runs in* $O(n^2)$ *time, so the overall run time of the decoding algorithm is* $O(n^4)$ *[1], [14].*

C. Case Study

*1) **Efficiency Test**:* To compare the efficiency of the unique indexing scheme and the synchronization strings (assuming the construction and decoding times can be neglected), we must consider operation at critical ε values: ε_L, the "equal length ε," ε_R, the "equal rate ε," and ε_P, the "minimum value ε."

Equal Length ε: There is some critical value ε_L where the total number of transmitted bits is equal in each scheme. Consider transmitting a message of length n symbols. The number of bits required to represent the n data symbols is given by $d(n)$, and the number of bits required to represent the n indexing symbols is given by $i(n)$. In either scheme, the required number of data bits $d(n)$ are equal, so ε_L is obtained by setting $i_U(n) = i_S(n)$:

$$i_U(n) = i_S(n)$$

Recall that each of the n indices in the unique indexing scheme requires $\log n$ bits, and that each of the n indices in the ε-synchronization string requires $\log|\Sigma_S| = \log\varepsilon_L^{-4}$ bits. Therefore:

$$n\log n = n\log\varepsilon_L^{-4} \longrightarrow \log n = \log\varepsilon_L^{-4}$$

Simplifying and rearranging:

$$n = \frac{1}{\varepsilon_L^4}$$

$$\therefore \varepsilon_L = n^{-1/4} \tag{1}$$

If a $\varepsilon < \varepsilon_L$-synchronization string is created, the unique indexing scheme requires fewer overall transmitted bits; instead, if $\varepsilon > \varepsilon_L$, then the synchronization string approach requires fewer overall bits.

Equal Rate ε: Similar to ε_L, there is some critical value ε_R where the rate of the overall insdel code R_S is equal in both schemes. Consider a transmission of length n using a base code C with rate R_C and alphabet Σ_C. Setting the net rates of both schemes equal to each other:

$$R_{SU} = R_{SS}$$

$$\frac{R_C}{\log n} = R_C\left(1 - \frac{\log|\Sigma_S|}{\log|\Sigma_C|}\right) \longrightarrow \frac{1}{\log n} = 1 - \frac{\log|\Sigma_S|}{\log|\Sigma_C|}$$

Simplifying:

$$\frac{\log|\Sigma_C|}{\log n} = \log|\Sigma_C| - \log|\Sigma_S|$$

$$\log|\Sigma_S| = \log|\Sigma_C| - \frac{\log|\Sigma_C|}{\log n}$$

$$\log|\Sigma_S| = \log|\Sigma_C|\left(1 - \frac{1}{\log n}\right)$$

$$\log\varepsilon_R^{-4} = \log|\Sigma_C|\left(1 - \frac{1}{\log n}\right)$$

$$\varepsilon_R^{-4} = 2^{\log|\Sigma_C|\left(1-\frac{1}{\log n}\right)}$$

$$\therefore \varepsilon_R = 2^{-\frac{\log|\Sigma_C|}{4}\left(1-\frac{1}{\log n}\right)} \tag{2}$$

To simplify, assume $|\Sigma_C| = 2^q$, and $n = 2^p$. Therefore:

$$\therefore \varepsilon_R = 2^{-\frac{q}{4}\left(1-\frac{1}{p}\right)} \tag{3}$$

If a $\varepsilon < \varepsilon_R$-synchronization string is created, then the rate R_S of the insdel code implemented using the unique indexing scheme will be larger; instead, if $\varepsilon > \varepsilon_R$, then the rate R_S of the insdel code implemented using the synchronization string approach will be larger.

Positive Rate Epsilon: The insdel code must have positive rate R_S; there is some minimum value ε_P that synchronization strings can use such that $R_{SS} > 0$:

$$R_C \left(1 - \frac{\log|\Sigma_S|}{\log|\Sigma_C|}\right) > 0 \iff \frac{\log|\Sigma_S|}{\log|\Sigma_C|} < 1$$

$$\log|\Sigma_S| < \log|\Sigma_C|$$

$$|\Sigma_S| < 2^{\log|\Sigma_C|}$$

$$\varepsilon_P^{-4} < 2^{\log|\Sigma_C|}$$

$$\frac{1}{\varepsilon^4} < 2^{\log|\Sigma_C|}$$

$$\therefore 2^{-\frac{1}{4}\log|\Sigma_C|} < \varepsilon_P \tag{4}$$

Using the simplifying assumption that $|\Sigma_C| = 2^q$:

$$\therefore 2^{-\frac{q}{4}} < \varepsilon_P \tag{5}$$

Therefore, to ensure codes with positive rate R_S, a synchronization string with $\varepsilon_P < \varepsilon$ is required.

All three of these equations can be combined to determine the regime where the synchronization string method is more efficient than the unique indexing method. We claim that if a synchronization string of length n is created with ε such that $\varepsilon_P < \varepsilon$ and $\varepsilon_L < \varepsilon_R \leq \varepsilon$, then for a given base code C with rate R_C and alphabet $|\Sigma_C| = 2^q$, the synchronization string is a more efficient indexing sequence than the unique indexing sequence. This claim logically follows from the observations made above; a synchronization string necessarily requires $\varepsilon_P < \varepsilon$ to ensure the rate of the insdel code $R_S > 0$. If $\varepsilon_L < \varepsilon_R$, then the desired overall rate R_S can be achieved using fewer transmitted bits with the synchronization string scheme than the unique indexing scheme. Therefore, provided a synchronization string is created according to these bounds, the synchronization string method will be more efficient than the unique indexing method.

A Reed-Solomon code with $|\Sigma_C| = 2^8$ was selected as the base code. Using this code, the critical ε parameters were plotted below, along with the region where the synchronization string is more advantageous than the unique indexing scheme for the same net rate R_S. Using this particular code, insdel codes implemented with synchronization strings with lengths $n \geq 113$ are in fact more efficient than insdel codes implemented with the unique indexing scheme.

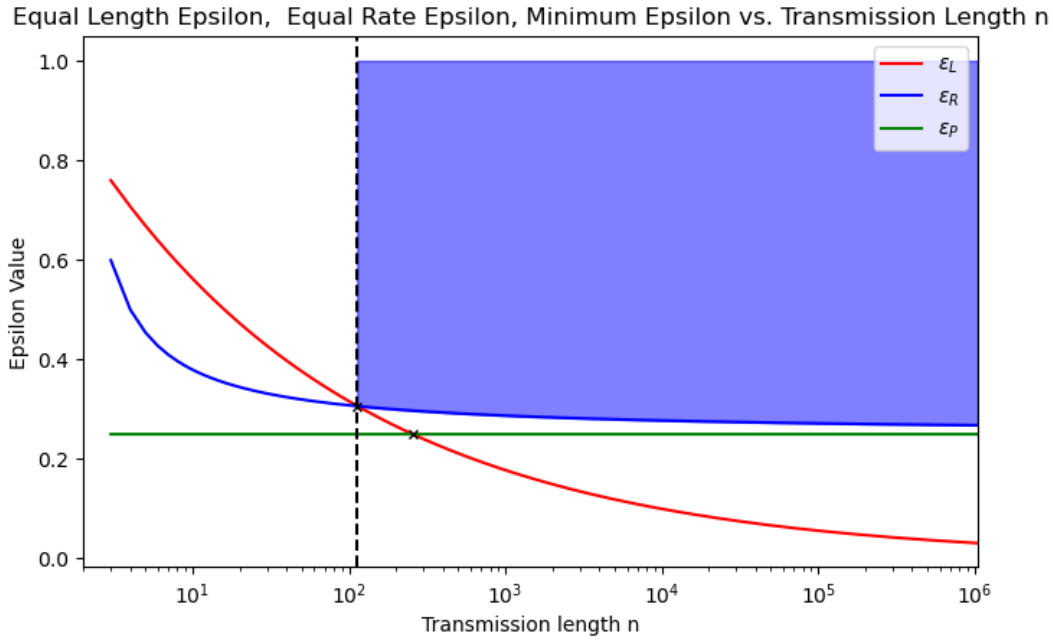

Fig. 9: *Critical* ε *parameters:* ε_L*, the equal length (red),* ε_R*, the equal rate (blue), and* ε_P*, the minimum (green). The region where synchronization strings have an advantage over unique indexing is shaded in blue. To find an advantageous synchronization string, simply select a transmission length* $n \geq 113$ *(the intersection point where* $\varepsilon_L \approx \varepsilon_R$*) and select any* ε *in the shaded region.*

2) ***Probability of Error Test****:* The number of erasures induced by the unique indexing decoder $\mathcal{D}_S$ is given as $h = f(n,\delta) \approx n\delta$. The approximation just accounts for any edge cases that may occur in the simulation; for simplicity, we can generally assume that $h = n\delta$. In this Case Study, a Reed-Solomon code was used as the base code $\mathcal{C}$; such a code can correct for $n-k$ erasures, where k is the number of message symbols encoded in block length n. If the maximum number of induced erasures is $h = n\delta$ and we want to guarantee the scheme can recover the original message with $p_{error} = 0$, then:

$$n - k \geq n\delta \Longrightarrow \delta \leq 1 - \frac{k}{n}$$

$$\therefore \delta_{t,U} \leq 1 - R_{\mathcal{C}} \tag{6}$$

Where $R_{\mathcal{C}} = \frac{k}{n}$ and δ_t is the maximum channel noise threshold for which it is guaranteed the scheme can recover the original message with $p_{error} = 0$. For a fixed $R_{\mathcal{C}} = \frac{k}{n}$, it is possible to operate in a channel with $\delta > \delta_t$, but in this regime, $p_{error} > 0$ - no guarantee can be made on the probability of errors in the recovered message.

Similar calculations can be carried out for the synchronization string method. In this case, $\mathcal{D}_S$ induces a maximum of $h = f(n,\delta,\varepsilon) \leq \left(\frac{5-\varepsilon}{1-\varepsilon}\right) n\delta$ erasures. Using the same Reed-Solomon base code, we calculate the threshold δ_t as:

$$n - k \geq \left(\frac{5-\varepsilon}{1-\varepsilon}\right) n\delta \Longrightarrow \delta \leq \left(\frac{1-\varepsilon}{5-\varepsilon}\right)\left(1 - \frac{k}{n}\right)$$

$$\therefore \delta_{t,S} \leq \left(\frac{1-\varepsilon}{5-\varepsilon}\right)(1 - R_{\mathcal{C}}) \tag{7}$$

As $\varepsilon \to 0$, $\delta_{t,S} \to \frac{1}{5}\delta_{t,U}$, so it may appear that the synchronization string scheme is less tolerant to channel noise than the unique indexing scheme. However, we must recall that δ_t represents the maximum channel noise where we can **guarantee** $p_{error} = 0$ if $\delta < \delta_t$. The additional "error overhead" in the synchronization case comes from Theorems 4.1 and 5.8 from [1]. The authors note that there are two contributions to h: the $n\delta$ insertions and deletions themselves, and $2k$ misdecodings stemming from the minimum RSD decoding algorithm. Haeupler

and Shahrasbi prove that the maximum number of misdecodings using the minimum RSD decoding algorithm is given by $k \leq \frac{2n\delta}{1-\varepsilon}$. In general, it is true that the upper bound on the number of induced erasures is higher in the synchronization string method than the unique indexing method, thus decreasing the channel noise threshold $\delta_{t,S}$ where it is guaranteed that $p_{error} = 0$. However, in practice, the actual number of misdecodings by the minimum RSD decoding algorithm is often much less than the upper-bound, enabling the synchronization string method to be effective over a wider range of noise levels, even above $\delta_{t,S}$. While there is no guarantee that $p_{error} = 0$ for $\delta > \delta_{t,S}$, in practice the probability of error is small enough to allow for greater effective use of the synchronization string method in channels with high noise levels.

The same Reed-Solomon code with $|\Sigma_C| = 2^8$, $R_C = \frac{1}{2}$, and block length $n = 32$ (therefore message length $k = 16$) was used for the probability of error tests. To equalize the overall rate of the insdel code R_S in both schemes (thus enabling meaningful comparisons of the probability of error), a synchronization string was created with $\varepsilon = \varepsilon_R \approx 0.32988$. Using these conditions, the noise threshold of the channel was set according to the unique indexing scheme with, $\delta_t = \frac{1}{2}$. $T = 50$ values of δ were chosen in evenly spaced intervals over the range $(0,1)$. At each noise level, $N = 100$ random messages of length $L = 500$ symbols were generated. Each message was parsed into chunks of size $k = 16$, encoded using the base Reed-Solomon coder, attached with the correct indexing sequence, and then transmitted over the insertion-deletion channel. The receiver will attempt to decode the data, first from the insertions and deletions and second from the induced erasures. If the original message is exactly recovered, then the transmission was considered successful; otherwise, the transmission is marked in error.

Two separate error models were considered:

- Fixed Error Model: For the channel noise δ and transmission length n, exactly $n\delta$ insertions and deletions were applied to the transmitted data. This model primarily serves as the benchmark.
- I.I.D. Error Model: The probability of applying a synchronization error at each index in the stream is δ; The expected number M of insertions/deletions is $E[M] = n\delta$.

The results of the simulations are summarized below:

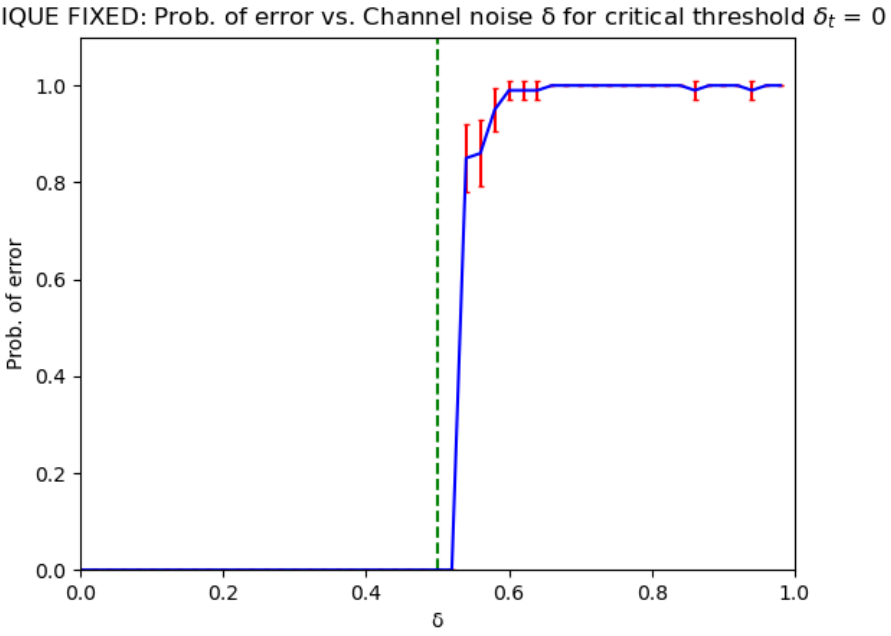

Fig. 10: *As expected, $p_{error} = 0$ for $\delta \leq \delta_t$; provided the base code C is properly designed with the required rate R_C, the unique indexing scheme can correct for all fixed number of insertions and deletions for channel noise $\delta \leq 1 - R_C$. While some successful decodes were possible for channel noise above the threshold, the probability of error quickly saturated at $p_{error} \approx 1$ for large δ.*

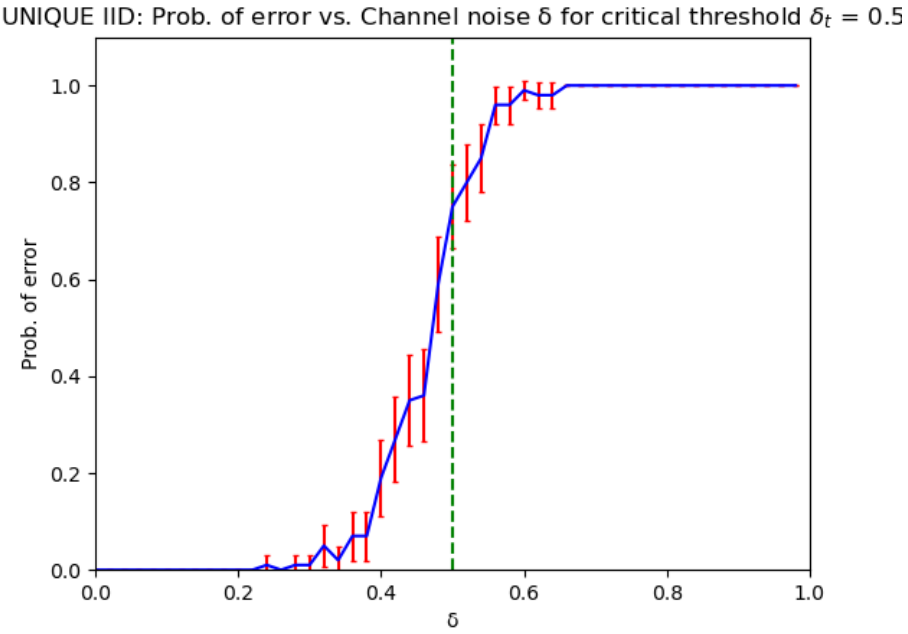

Fig. 11: *For $\delta < \delta_t$, we see that $p_{error} > 0$; due to the I.I.D. nature of introduced insertions/deletions, there is a finite probability that the number of induced erasures h exceeds the error-correcting capabilities of the base code $\mathcal{C}$, even for $\delta < \delta_t$. The probability of error is still large for $\delta > \delta_t$. In this error model, we see that the unique indexing scheme is largely ineffective; for $\delta < \delta_t$, the probability of error is much higher than the fixed-error benchmark, and for $\delta > \delta_t$ the curve looks virtually identical to the fixed-error benchmark.*

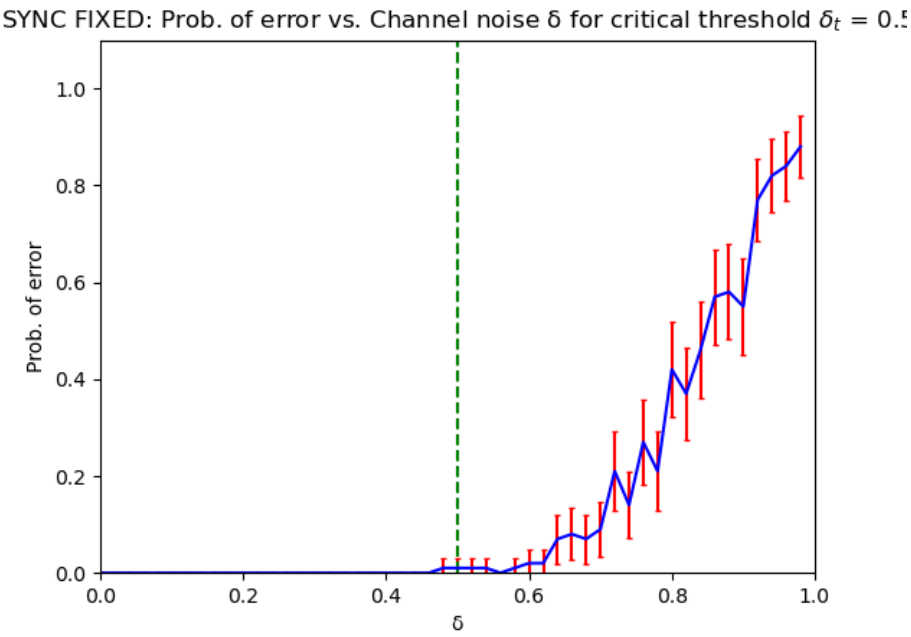

Fig. 12: *For* $\varepsilon = 0.32988$*, the threshold noise level is given as* $\delta_{t,S} = 0.07175$*; below this threshold,* $p_{error} < 0$*, as expected. This is not too difficult to accomplish, given such a low noise level. However, as expected, the true strength of the minimum RSD decoder is observed. While the algorithm may require a larger upper bound on the number of induced erasures, in practice - even in extremely high noise regimes - the number of erasures induced by the minimum RSD decoder is often significantly smaller than the upper bound, well within the error-correcting capabilities of the base code* C*. The probability of error still increases close to* 1 *for large* δ*, as expected, but compared to the unique indexing scheme, the synchronization string is significantly more resilient to errors.*

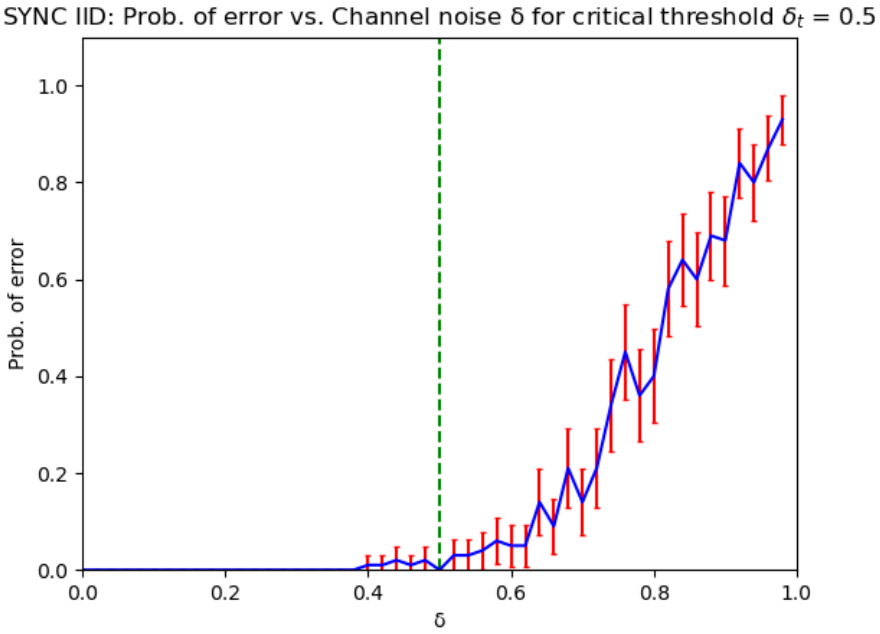

Fig. 13: *Changing the error model from fixed to I.I.D. did not have a dramatic impact on the probability of error curve, again largely due to the successful properties of the minimum RSD decoding algorithm. In some cases, the probability of error was slightly higher than in the fixed error benchmark, but overall the curves were very similar. This suggests that synchronization strings are more resilient and robust (operating in different error models) than the unique indexing scheme.*

*3) **Performance**:* $N = 100$ ε_R-synchronization strings of lengths up to $n = 64$ were constructed. A single synchronization string of length $n = 128$ was constructed. At each length, the average construction time was computed. These strings were then used in a fixed-error transmission. For $n < 128$, $T = 100$ transmissions were simulated, and the index decoding time measured. For $n = 128$, 15 transmissions were simulated, and the index decoding time measured. The results are summarized in the figures below:

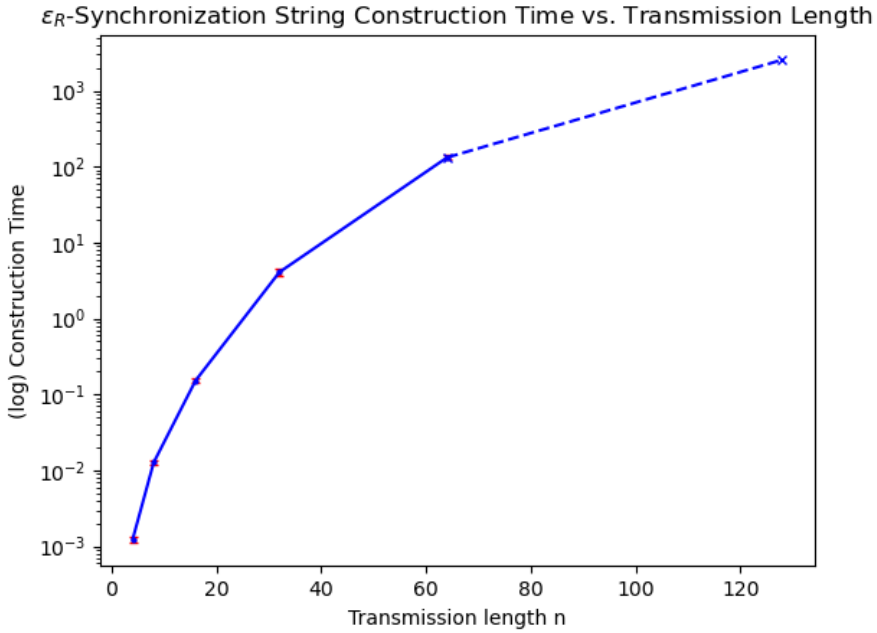

Fig. 14: *Average construction times for ε_R-synchronization strings of length $n = 4, 8, 16, 32, 64, 128$. While the average construction times may be somewhat acceptable for small n, they become increasingly impractical to generate for large $n \geq 128$. Due to the large construction times, only a single synchronization string of length $n = 128$ could be constructed, so the presented value is given as an extremely broad estimate. However, an interesting property of synchronization strings is that only one string (for a given length n and given ε) is ever required for indexing purposes. Provided the length of the transmission is $n = 128$, and the application was designed for the parameter ε_R for $n = 128$, the application can continuously reuse the same synchronization string as the indexing sequence S in each transmission. This suggests that precomputing synchronization strings in advance of deployment in the field is a viable strategy to virtually eliminate large construction time during run-time of the application.*

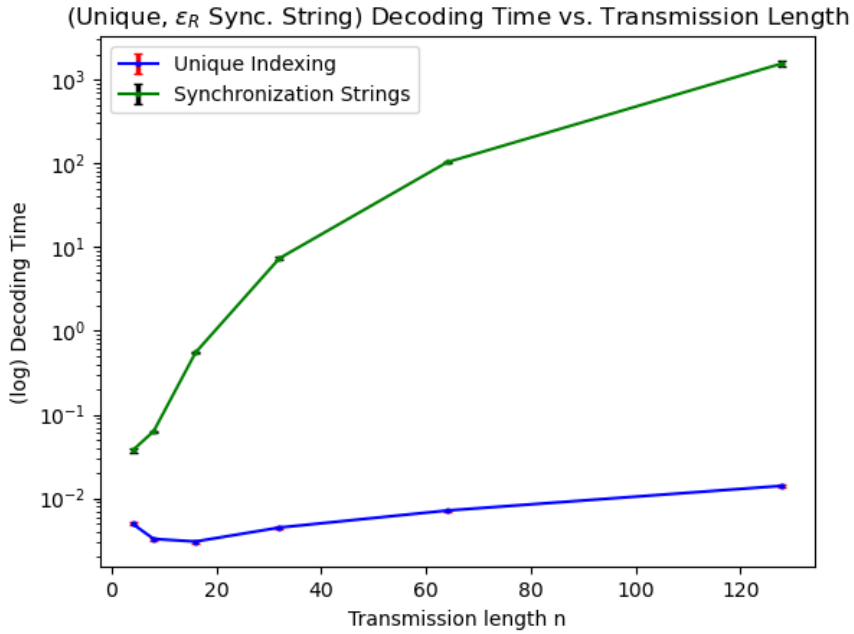

Fig. 15: *Average decoding times for ε_R-synchronization strings and unique indexing sequences of length $n = 4, 8, 16, 32, 64, 128$. Similar behavior to the construction times are observed; the unique indexing decoding method is several orders of magnitude faster than the synchronization string decoding method. Like the construction case, synchronization strings become impractical to decode for large $n \geq 128$. However, the minimum RSD decoding algorithm is naturally a good candidate for parallelization: RSD calculations for each substring $S'[1, j]$ can be carried out independently and in parallel. While it may not be possible to achieve decoding times at the scale of the unique indexing case, it should be possible to significantly reduce the decoding time such that synchronization strings can be deployed in more practical settings.*

D. Tables and Equations

Name	Equation
Edit Distance	$ED\{S,S'\}$ = Min. number of insertions and deletions required to convert S into S' [1]
Relative Suffix Distance	$RSD\{S,S'\} = \max_{k>0} \frac{ED\{S(\lvert S\rvert-k,\lvert S\rvert], S'(\lvert S'\rvert-k,\lvert S'\rvert]\}}{2k}$
ε-synchronization Property	$ED\{S[i,j), S[j,k)\} > (1-\varepsilon)(k-i)\ \forall 1 \leq i < j < k \leq n+1$
Equal Length ε_L	$\varepsilon_L = n^{-1/4}$
Equal Rate ε_R	$\varepsilon_R = 2^{-\frac{\log\lvert\Sigma_C\rvert}{4}\left(1-\frac{1}{\log n}\right)} = 2^{-\frac{q}{4}\left(1-\frac{1}{p}\right)}$, if $\lvert\Sigma_C\rvert = 2^q, n = 2^p$
Minimum Epsilon ε_P	$2^{-\frac{1}{4}\log\lvert\Sigma_C\rvert} = 2^{-\frac{q}{4}} < \varepsilon_P$, if $\lvert\Sigma_C\rvert = 2^q$

TABLE I: *Table of Equations*

Field	Unique Indexing	Synchronization String
Base Code	Any $\mathcal{C}$	Any $\mathcal{C}$
Number of Induced Erasures $h = f(n,\delta,\varepsilon)$	$f(n,\delta) = n\delta$	$f(n,\delta,\varepsilon) \leq \left(\frac{5-\varepsilon}{1-\varepsilon}\right) n\delta$
Noise Threshold δ_t where guaranteed $p_{error} < 0$	$1 - R_C$	$\left(\frac{1-\varepsilon}{5-\varepsilon}\right)(1-R_C)$
Insdel Code Rate R_S	$\frac{R_C}{\log n}$	$R_C\left(1 - \frac{\log\lvert\Sigma_S\rvert}{\log\lvert\Sigma_C\rvert}\right)$
Indexing Alphabet Size $\lvert\Sigma_S\rvert$	n	$\Theta\left(\varepsilon^{-4}\right)$
Index Construction Time $T_{S_{ENC}}$	$O(n)$	$O(n^5)$
Index Decoding Time $T_{S_{DEC}}$	$O(n^2)$	$O(n^4)$

TABLE II: *Summary of properties for each scheme*

REFERENCES

[1] B. Haeupler and A. Shahrasbi, "Synchronization Strings: Codes for Insertions and Deletions Approaching the Singleton Bound," *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 33–46, Jun. 2017, arXiv: 1704.00807. [Online]. Available: http://arxiv.org/abs/1704.00807

[2] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals." *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, feb 1966, doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[3] J. Ullman, "Near-optimal, single-synchronization-error-correcting code," *IEEE Transactions on Information Theory*, vol. 12, no. 4, pp. 418–424, Oct. 1966. [Online]. Available: http://ieeexplore.ieee.org/document/1053920/

[4] L. Schulman and D. Zuckerman, "Asymptotically good codes correcting insertions, deletions, and transpositions," *IEEE Transactions on Information Theory*, vol. 45, no. 7, pp. 2552–2557, Nov. 1999. [Online]. Available: http://ieeexplore.ieee.org/document/796406/

[5] A. Bours, "Construction of fixed-length insertion/deletion correcting runlength-limited codes," *IEEE Transactions on Information Theory*, vol. 40, no. 6, pp. 1841–1856, Nov. 1994. [Online]. Available: http://ieeexplore.ieee.org/document/340459/

[6] H. Mercier, V. Bhargava, and V. Tarokh, "A survey of error-correcting codes for channels with symbol synchronization errors," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 1, pp. 87–96, 2010. [Online]. Available: http://ieeexplore.ieee.org/document/5415865/

[7] E. Tanaka and T. Kasai, "Synchronization and substitution error-correcting codes for the Levenshtein metric," *IEEE Transactions on Information Theory*, vol. 22, no. 2, pp. 156–162, Mar. 1976. [Online]. Available: http://ieeexplore.ieee.org/document/1055532/

[8] V. Guruswami and R. Li, "Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes," in *2016 IEEE International Symposium on Information Theory (ISIT)*. Barcelona, Spain: IEEE, Jul. 2016, pp. 620–624. [Online]. Available: http://ieeexplore.ieee.org/document/7541373/

[9] "Reed-solomon codes for coders." [Online]. Available: https://en.wikiversity.org/wiki/Reed-Solomon_codes_for_coders

[10] K. Cheng, B. Haeupler, X. Li, A. Shahrasbi, and K. Wu, "Synchronization Strings: Efficient and Fast Deterministic Constructions over Small Alphabets," *arXiv:1803.03530 [cs, math]*, Mar. 2018, arXiv: 1803.03530. [Online]. Available: http://arxiv.org/abs/1803.03530

[11] J. Leahy, D. Touchette, and P. Yao, "Quantum Insertion-Deletion Channels," *arXiv:1901.00984 [quant-ph]*, Jan. 2019, arXiv: 1901.00984. [Online]. Available: http://arxiv.org/abs/1901.00984

[12] I. Shomorony and R. Heckel, "DNA-Based Storage: Models and Fundamental Limits," *arXiv:2001.06311 [cs, math]*, Jan. 2020, arXiv: 2001.06311. [Online]. Available: http://arxiv.org/abs/2001.06311

[13] T. Anker, R. Cohen, and D. Dolev, "Transport Layer End-to-End Error Correcting," p. 9, 2004.

[14] B. Haeupler and A. Shahrasbi, "Synchronization strings: Codes for insertions and deletions approaching the singleton bound," *CoRR*, vol. abs/1704.00807, 2017. [Online]. Available: http://arxiv.org/abs/1704.00807