RECIPE: Rateless Erasure Codes Induced by Protocol-Based Encoding

Jingfan Meng
Georgia Tech
jfmeng@gatech.edu

Ziheng Liu
University of Utah
ziheng.liu@utah.edu

Yiwei Wang
Georgia Tech
ywang3607@gatech.edu

Jun Xu
Georgia Tech
jx@cc.gatech.edu

***Abstract*—LT (Luby transform) codes are a celebrated family of rateless erasure codes (RECs). Most of existing LT codes were designed for applications in which a centralized encoder possesses all message blocks and is solely responsible for encoding them into codewords. Distributed LT codes, in which message blocks are physically scattered across multiple different locations (encoders) that need to collaboratively perform the encoding, has never been systemically studied before despite its growing importance in applications. In this work, we present the first systemic study of LT codes in the distributed setting, and make the following three major contributions. First, we show that only a proper subset of LT codes are feasible in the distributed setting, and give the sufficient and necessary condition for such feasibility. Second, we propose a distributed encoding protocol that can efficiently implement any feasible code. The protocol is parameterized by a so-called action probability array (APA) that is only a few KBs in size, and any feasible code corresponds to a valid APA setting and vice versa. Third, we propose two heuristic search algorithms that have led to the discovery of feasible codes that are much more efficient than the state of the art.**

I. Introduction

Rateless erasure codes (REC) [1] are a powerful tool for reliable data transmission. LT (Luby transform) codes [2] are the best-known family of REC. LT codes are attractive for network applications, because they have both high coding efficiency and low decoding time complexity (using the peeling algorithm [3]). Most existing LT codes were designed for applications in which a centralized encoder possesses all message blocks and is solely responsible for encoding them into codewords. However, the past two decades has seen some applications in which the message blocks are physically scattered across multiple different locations (encoders) that need to collaboratively perform the encoding.

In this work, we perform the first systemic study of LT codes in the distributed setting, by posing three research questions. Our starting question is "Does the distributed setting impose certain additional constraints that make some LT codes not realizable (feasible)?" Since the answer to this question is yes, as we will explain shortly, it leads to two more questions. Our second question is "Can we design an distributed encoding protocol that can be parameterized to realize (implement) any feasible LT code, and has low computational and storage overheads for the encoders?" Our third question is "Can we find good codes (with high coding efficiency) within this restricted (feasible) family?"

In this work, we make three major contributions by definitively answering these three questions, respectively. Our first contribution is to show that only a proper subset of LT codes are feasible in the distributed setting, and to give the sufficient and necessary condition for an LT code to be feasible. Our second contribution is to propose a distributed encoding protocol that can efficiently implement any feasible code. Hence, we call both this protocol, and the family of codes it generates, RECIPE (Rateless Erasure Codes Induced by Protocol-based Encoding). We say a code is RECIPE-feasible if it belongs to this family. The ultimate goal of the RECIPE coding theory is to discover RECIPE-feasible codes that can achieve high coding efficiencies. However, to search for such codes appears challenging, as we will elaborate in Section V. Our third contribution is two heuristic search algorithms[1] that have led to the discovery of RECIPE-feasible codes that are much more efficient than the state of the art [4].

II. Background and Formulation

A. Centralized LT codes and XOR Degree Distributions

In this section, we provide a brief introduction to centralized LT coding concepts, terms, and notations. Let U be the set of message blocks to be encoded (for transmission) and $k = |U|$; k is called block size in the literature. Each LT codeword is an *XOR* sum of d distinct, randomly selected, message blocks. We refer to this set (of message blocks) as the XOR-set in the sequel. This random selection is required to be uniform in the sense that, for any $i \leq k$, all $\binom{k}{i}$ ways (of selecting i distinct message blocks from U to XOR together) must be equally likely. We refer to this requirement as the uniformity condition in the sequel. This d, which is called the *XOR degree*, is in general a random variable. We call the probability distribution of d XOR *degree distribution* (XDD) and denote it as $\vec{\mu}$ in this paper. Thanks to the uniformity requirement, any LT coding scheme is uniquely determined and hence defined by its XDD $\vec{\mu}$. Throughout this paper, we write a rightward arrow on the top of $\vec{\mu}$ to emphasize that it is a vector of k scalars, in which the i^{th} scalar is denoted by $\mu(i)$.

B. Path Tracing: Our Distributed LT Coding Problem

In this section, we describe the path tracing problem that our RECIPE coding theory is designed to tackle. Probabilistic

[1]We will release all source codes and resulting XDD distributions on https://cc.gatech.edu/home/jx.

in-band network telemetry (PINT) is an emerging protocol framework for real-time data center network (DCN) measurement and monitoring [4], [5]. An important PINT task is path tracing [6]–[8]. In path tracing, each participating switch (router) probabilistically encodes its (switch) ID into a dedicated "PINT field" contained in each packet transiting through the switch; this field is typically short (say no more than 16 bits) to keep the bandwidth overhead of the path tracing operation small. The goal of path tracing is for the destination host of a flow of packets to recover the entire network path this flow had traversed, from the codewords ("PINT field" values) of these packets.

Formally, path tracing can be modeled as a distributed LT coding problem as follows. Consider a path tracing instance in which a source node SRC sends a flow of packets to a destination node DST, following a path that contains switches 1, 2, $\cdots$, k in that order. This instance corresponds to a coding instance, in which the k switches are the encoders that each possesses a message block (which is its own switch ID), and DST is the decoder. The "PINT field" in each packet is a codeword-in-progress when the packet traverses along the path from SRC to DST. The distributed coding problem is how these k switches should collaboratively encode each such codeword so that DST can recover the entire path from as few packets (codewords therein) as possible (i.e., achieve high coding efficiency), which is important for such an LT code to be useful in a data center network environment in which the vast majority of flows contain no more than several packets.

In this instance, each switch i along the path can "do something" to a codeword-in-progress C only during the packet's "brief stay" at i. In other words, whatever i decides to do to C, it is a one-shot online decision. As shown in [4], there are only three conceivable LT coding actions i can do to C (probabilistically): `Add` (XOR its ID M_i with C), `Skip` (do nothing to C), and `Replace` (C with M_i). These three actions are however sufficiently expressive, since it will become intuitive to readers that they themselves do not constrain in any way how "large and rich" the RECIPE-feasible family (of codes) can be.

Rather, what makes this coding problem challenging and "shapes" the RECIPE-feasible family is the following constraint imposed by the data center network environment. The constraint is that every switch (encoder) has to be stateless and "weightless" in the sense it performs the same extremely simple ("weightless") LT encoding processing on every transiting packet without *consciously knowing* which path (coding instance) this packet travels (belongs to) or the path length. This constraint is necessary for a path tracing operation to incur minimal systems overheads at data center switches, since at any moment a switch may be on the paths of millions of different source-destination flows, each of which corresponds to a different coding instance. Under this constraint, every switch has to probabilistically perform `Add`, `Skip`, or `Replace`, with the same probability parameter settings (that we will elaborate in Section III), on each packet (the codeword-in-progress therein) transiting through it, independently.

We now elaborate, using the coding instance above, on the aforementioned "not knowing the path length", since it is arguably the most consequential part of the stateless constraint. In general, any switch i, $i = 1, 2, \cdots, k$, does not know the exact value of k (the block size of this instance) when processing a packet (codeword-in-progress) belonging to this instance. To be more precise, switch i knows its "position" i (from the time-to-live (TTL) field in the IP header of the packet), but not $k-i$ ("how far away" the packet is from DST). This constraint implies that, at any switch i ($1 \leq i \leq k$) in this instance, the codewords-in-progress (contained in the packets of the flow) after being processed by switches 1 through i must be (the realizations of) a valid code in the sense the (XOR-set distribution of the) code satisfies the uniformity condition and hence can be characterized by an XDD that we denote as $\vec{\mu}_i$. This is because any switch i could be the last switch in another coding instance Γ (of length i), and in this case the codewords-in-progress that switch i (in instance Γ) transits are the final codewords. This constraint also implies that a (parameterized) RECIPE protocol should induce the same XDD $\vec{\mu}_i$ for all instances (in the network) of size (length) i, since the probabilistic "decision logic" (whether the action should be `Add`, `Skip`, or `Replace`) at switches 1 through i in all such instances are identical due to the stateless constraint. Hence a RECIPE protocol, which is run concurrently by numerous coding instances of different path lengths in a data center network, generally induces a sequence of XDDs $\vec{\mu}_1, \vec{\mu}_2, \cdots, \vec{\mu}_K$ (one for each path length), where K is the maximum path length (i.e., diameter) in the network. Our aforementioned first contribution is to discover the necessary and sufficient condition these K XDDs need to satisfy for their "concatenation" to become a RECIPE-feasible code.

Throughout this work, for ease of presentation, we assume that the PINT field in each packet has the same length as the ID of a switch, and that an LT codeword is the XOR sum of switch IDs. The PINT paper [4] has proposed several solutions for cases in which the PINT field is shorter, such as hash-compressing the switch IDs or fragmenting a switch ID across multiple packets.

C. PINT: The State of the Art Path Tracing Code

The only prior work on the topic of distributed LT coding for path tracing is PINT [4], which proposed a reasonably good code, but did not develop any theory. In comparison, this work gets to the bottom of the problem, explores the entire design space, and reaps the reward of discovering much more efficient codes than the PINT code.

The PINT code can be considered a linear combination of two (what we now call) RECIPE codes (XDD sequences). In the first code, each packet, when arriving at its destination, carries in its codeword a uniformly and randomly chosen switch ID along its path. This code is produced by switches performing reservoir sampling. The second code is produced by every switch *XOR*-ing its switch ID to the codeword contained in an arriving packet with a fixed probability p. In the

resulting code (XDD sequence), each $\vec{\mu}_k$, $k = 1, 2, \cdots, K$, is precisely $Binomial(k, p)$. Neither code is efficient. PINT [4] uses a linear combination of these two inefficient codes that is, surprisingly, much more efficient than both.

III. RECIPE AND ITS VARIANT

In this section, we first present (in Section III-A) the baseline RECIPE protocol, called RECIPE-d, that requires each codeword-in-progress C to be accompanied with the value of d, the current XOR-degree of C. This requirement increases the coding overhead by a few (say 6) bits per packet (as $d < 64$ in any current communication network). This extra coding overhead can be eliminated using a streamlined variant of RECIPE that we call RECIPE-t and present in Section III-B. RECIPE-t can induce any RECIPE-feasible code approximately but accurately, at the tiny cost of requiring each encoder (switch) to store a small (no more than 1MB in size) precomputed table. RECIPE-d and RECIPE-t are the second aforementioned contribution of this work.

A. Degree-Based RECIPE (RECIPE-d)

Algorithm 1: RECIPE-d protocol by switch i.

1 Retrieve hop count i, *XOR* degree d, and codeword C from pkt;
2 $\nu \leftarrow h(i, pkt)$; // $\nu \in [0, 1)$
3 **if** $\nu < p_A(i, d)$ **then**
4 | $C \leftarrow C \oplus M$; // Add action
5 **else if** $\nu < p_A(i, d) + p_R(i, d)$ **then**
6 | $C \leftarrow M$; // Replace action
7 **else**
8 | Do nothing; // Skip action
9 Update d accordingly;
10 Write d and C back to the packet.

Alg. 1 shows how a switch whose ID is M processes a packet pkt (the codeword-in-progress C therein) transiting through it, using the RECIPE-d protocol (that is run by all switches with the same parameter setting). As shown in Lines 3 through 9, the switch performs one of the three aforementioned actions (`Add`, `Skip`, and `Replace`) on C with probability $p_A(i, d)$, $p_S(i, d)$, and $p_R(i, d)$ respectively. Here i is how far (in number of hops) this switch is away from the SRC of pkt, which as mentioned earlier can be inferred from pkt's TTL; d is the current XOR-degree of C (that RECIPE-d "pays" to know as mentioned earlier). As such, the RECIPE-d protocol is parameterized by the 2D array $(p_A(i, d), p_S(i, d), p_R(i, d))$, $i = 0, 1, \cdots, K$ and $d = 1, 2, \cdots, i - 1$, that we call the action probability array (APA). How to set (probability values in) APA so that the resulting RECIPE-d protocol induces a valid and a good code will be studied in Section IV and Section V, respectively. RECIPE-d is stateless since the (random) action the switch performs on pkt depends only on i and d, but not on the flow (coding instance) pkt belongs to.

In any LT coding scheme, to decode a set of codewords, the host (in our case the DST) must know the XOR-set of each codeword in the set. Alg. 1 uses a standard (in computer science) derandomization technique called global hashing that allows the DST of pkt to recover the XOR-set of C as follows. A global hash function $h(\cdot, \cdot)$ is shared among all switches and hosts in the network. As shown in Lines 2 through 9, the exact realized action this switch performs on pkt is determined by the hash value $\nu = h(i, pkt)$. When pkt reaches DST, DST can infer the XOR-set of C therein from the hash values $h(i, pkt)$, $i = 1, 2, \cdots$, that DST itself can compute.

B. Table-based RECIPE (RECIPE-t)

Consider a hypothetical path of maximum possible length K and a packet pkt that travels down the hypothetical path to its DST. We define the following (random) action vector $\overrightarrow{act} \triangleq (act(1), act(2), \cdots, act(K))$, where each $act(i)$, $i = 1, 2, \cdots, K$, is the random action that switch i performs on pkt (the codeword C therein). Recall that in RECIPE-d, switch i realizes only $act(i)$ according to the hash value $\nu = h(i, pkt)$ and the APA that parameterizes the protocol. The idea of RECIPE-t is to let every switch store an (identical) copy of a precomputed (via Monte-Carlo simulation of RECIPE-d) action vector sample table (AVST) whose rows are independent realizations of the random vector $\overrightarrow{act}$. Suppose the AVST has L rows (independent samples) that we denote as $\overrightarrow{act}_1$, $\overrightarrow{act}_2$, $\cdots$, $\overrightarrow{act}_L$. When pkt travels down its path, all switches along the path collaboratively sample one uniformly random (across $[L] \triangleq \{1, 2, \cdots, L\}$) row l in AVST (i.e., $\overrightarrow{act}_l$); and for $i = 1, 2, \cdots$, switch i performs $act_l(i)$ on the codeword contained in the packet. This sampling (of l) is done collaboratively using a different global hash function $g(pkt)$ (than $h(i, pkt)$). In theory, when L tends to infinity, the (approximate) XDD sequence RECIPE-t induces converges to the actual XDD it tries to "simulate". In practice, $L = O(10^4)$ is large enough to achieve a very close approximation, as will be shown in Figure 2.

IV. RECIPE-FEASIBLE XDD SEQUENCES

In this section, we state the first aforementioned contribution of this paper: the sufficient and necessary condition for an XDD sequence to be RECIPE-feasible. The sufficiency proof also explains how APA should be set to induce a RECIPE-feasible XDD sequence. The RECIPE coding theory in this and the next two sections will be developed for RECIPE-d, with the understanding that we can approximate any parameterized (by APA) RECIPE-d using its streamlined variant RECIPE-t.

To begin with, we introduce a notation that makes our presentation easier. Consider any XDD $\vec{\mu}_i$ in the XDD sequence. Recall that the uniformity condition means that for any $d \leq i$, the probability for every size-d subset of $[i]$ to be the XOR-set of C is identical. We denote this probability by $q_i(d)$, which is equal to $\mu_i(d)/\binom{i}{d}$ by definition. Throughput this section, we denote an XDD sequence by $\vec{q}_1, \vec{q}_2, \cdots, \vec{q}_K$ instead of $\vec{\mu}_1, \vec{\mu}_2, \cdots, \vec{\mu}_K$.

The following theorem shows that the family of RECIPE-feasible codes corresponds to a $(K(K+1)/2)$-dimensional polytope bounded by the following linear constraints (facets).

Theorem 4.1: An XDD sequence $\vec{q}_1, \vec{q}_2, \cdots, \vec{q}_K$ is RECIPE-feasible if and only if for any $2 \leq i \leq K, 1 \leq d \leq i-1$, it holds that

$$q_{i-1}(d) \geq q_i(d) + q_i(d+1). \tag{1}$$

The necessity part of Theorem 4.1 is proved in Appendix A in [9], and its sufficiency part follows from the following APA designation that instantiates any RECIPE-feasible XDD sequence $\vec{q}_1, \vec{q}_2, \cdots, \vec{q}_K$.

- For $i = 1$, the first switch always replaces the (initially empty) codeword by its ID, like in PINT. In other words, we let $p_A(1,0) = 0$, $p_S(1,0) = 0$, and $p_R(1,0) = 1$.
- For any $2 \leq i \leq K$ and $1 \leq d \leq i-1$, we let

$$\begin{aligned} p_A(i,d) &= q_i(d+1)/q_{i-1}(d), \\ p_S(i,d) &= q_i(d)/q_{i-1}(d), \\ p_R(i,d) &= 1 - p_A(i,d) - p_S(i,d). \end{aligned} \tag{2}$$

In the interest of space, we prove in Appendix B in [9] that the APA entries in (2) are well-defined for any RECIPE-feasible XDD sequence, and that a RECIPE-d protocol thus parameterized satisfies the uniformity condition and induces the XDD sequence $\vec{q}_1, \vec{q}_2, \cdots, \vec{q}_K$.

V. Search for Good XDD Sequences

It is very challenging to discover efficient codes in the RECIPE-feasible family for two reasons. First, the search for good codes has to work with the aforementioned $(K(K+1)/2)$-dimensional RECIPE-feasible polytope. Second, for a distributed LT code (XDD sequence) $\vec{\mu}_1, \vec{\mu}_2, \cdots, \vec{\mu}_K$ to be considered good (in terms of coding efficiency), every $\vec{\mu}_k$, $k = 1, 2, \cdots, K$, needs to be good, because if $\vec{\mu}_{k^*}$ for a certain path length k^* is bad, then all coding instances of path length k^* have low efficiency.

In this section, we propose two heuristic algorithms for searching for good RECIPE-feasible codes. The first algorithm, called HRS and to be described in Section V-A, searches over the entire RECIPE-feasible polytope. The second, called QPS and to be described in Section V-B, searches over a much smaller polytope called invariant (RECIPE-feasible) sequences, but is much more computationally efficient in exploring the smaller polytope. As a result, when K is large (say in hundreds), only QPS can output good codes (on every k) "in due time". We will plot the XDD of a "good" RECIPE code found by QPS in Appendix D in [9].

A. Heuristic Reversed Search (HRS)

Our first search algorithm, called heuristic reversed search (HRS), is to greedily solve a series of K subproblems that each has $O(K)$ variables to work with. The idea is to find good XDD's hop-by-hop in the reversed order (from last to first) while conforming to (1). We start with an XDD $\vec{\mu}_K$ at the last hop. The default choice is Robust Soliton [2] due to its high coding efficiency. Then, for $i = K, K-1, \cdots, 2$, we iteratively search for a good XDD $\vec{\mu}_{i-1}$ (for one hop earlier) in the RECIPE-feasible region (that satisfies (1) under current i and XDD $\vec{\mu}_i$). It is straightforward to show that every subproblem thus formulated is feasible.

HRS is reasonably computationally efficient for K values that are not too large (say $K \leq 128$). However, since the search for $\vec{\mu}_{i-1}$ depends on $\vec{\mu}_i$, $\vec{\mu}_{i-1}$ "inherits" any coding inefficiency of $\vec{\mu}_i$. As a result, when K is larger than 100 or so, $\vec{\mu}_k$ found by HRS are not very efficient except when k gets close to (the last hop) K.

B. Quadratic Programming Search (QPS)

Our second scheme, called quadratic programming search (QPS), searches over only invariant (XDD) sequences (defined next) in the RECIPE-feasible polytope.

Definition 5.1: An XDD sequence is *invariant* (at each hop) if and only if for every XOR degree $d = 1, 2, \cdots, K-2$, $\mu_{d+1}(d) = \mu_{d+2}(d) = \cdots = \mu_K(d)$.

By this definition, every invariant XDD sequence is fully parameterized by the K scalars in $\vec{\mu}_K$ (thus we drop the subscript K in the following theorem). Furthermore, as a direct corollary of Theorem 4.1, the following theorem shows that the subspace of XDD sequences that are both *invariant and RECIPE-feasible* is a K-dimensional polytope.

Theorem 5.2: An invariant XDD sequence is RECIPE-feasible if and only if for every $d = 1, 2, \cdots, K-2$, $\mu(d) \geq (d+1)/d \cdot \mu(d+1)$.

Example 5.3: It is not hard to verify that the truncated (at $k = 1, 2, \cdots, K$) Soliton distributions (see [2]), when concatenated into an XDD sequence, is neither invariant nor RECIPE-feasible. However, the following truncated XDDs $\vec{\mu}_k$, $k = 1, 2, \cdots, K$, when concatenated into an XDD sequence, is both invariant and RECIPE-feasible. We call this XDD sequence *Shifted Soliton*, since each $\vec{\mu}_k$ is a "cyclic shift" of the truncated (at k) Ideal Soliton. We accidentally discovered Shifted Soliton, which in turn inspired us to propose QPS (to search for even better invariant sequences using computer).

$$\begin{aligned} &\mu_k(d) = 1/[d(d+1)],\ d = 1, 2, ..., k-1. \\ &\mu_k(k) = 1/k. \end{aligned}$$

Since QPS searches (using a quadratic programming procedure as we will describe in Appendix C in [9], which gives QPS its name) over a K-dimensional "sub-polytope" (of invariant sequences) of the $(K(K+1)/2)$-dimensional RECIPE-feasible polytope, it is much more computationally efficient than HRS when K is large (say $K > 100$). Luckily, many good XDD sequences still exist within this "sub-polytope", which can be (relatively) rapidly found by QPS.

VI. Evaluation

In this section, we first compare SS (Shifted Soliton) and (codes discovered by) HRS and QPS, against PINT, the state-of-the-art distributed LT code. Then, we show that RECIPE-t achieves a similar coding efficiency as RECIPE-d given a moderately large AVST (no more than 1MB). We measure coding efficiency by the average number of codewords needed

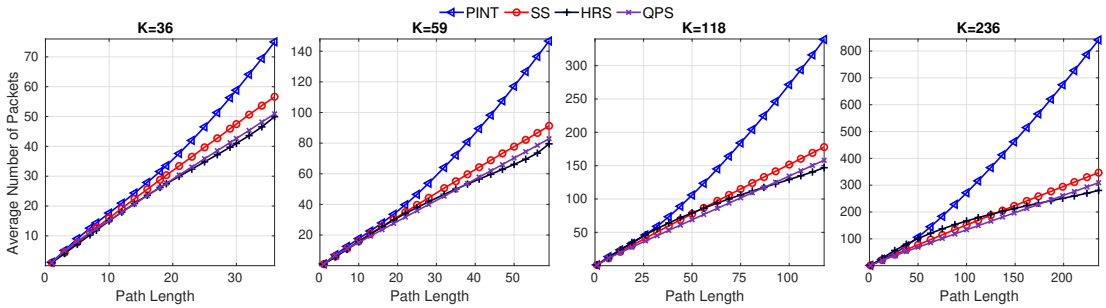

Fig. 1. RECIPE codes vs PINT in terms of coding efficiency.

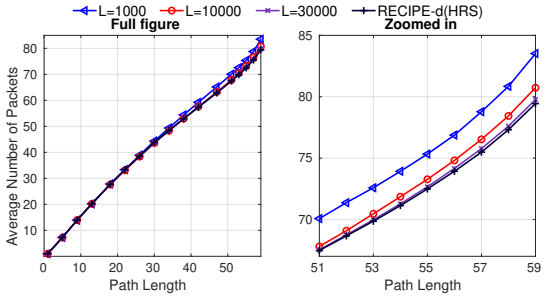

Fig. 2. RECIPE-t compared against RECIPE-d.

to completely decode a path (of k hops). According to our experiments, the comparison results are roughly the same when other metrics such as the 99% quantile are used (see Appendix D in [9]).

Comparison with PINT: We use the following two typical values of network diameter K from the evaluation of PINT in [4]: $K = 36$ (US Carrier dataset), and $K = 59$ (Kentucky Datalink dataset). The other two diameter values, $K = 118$ and $K = 236$, result from fragmenting each switch ID in Kentucky Datalink to 2 and 4 message blocks, respectively.

As shown in Figure 1, SS and QPS outperform PINT consistently (i.e., for every possible path length) from 1 to 236. HRS, however, underperforms PINT at small k values when K is 118 or 236 due to the (cumulative) "inherited" coding inefficiency (while searching backward from K) we mentioned earlier. When the path length $k = K$, SS, HRS and QPS all outperform PINT significantly, by 24.5% to 58.8%, 33.4% to 66.7%, and 32.2% to 63.2%, respectively. QPS consistently outperforms SS, which is expected since SS is discovered by our "naked eye" in the same "subpolytope" whereas QPS is by the computer. In the three experiments using Kentucky Datalink (in which $K \geq 59$), HRS outperforms QPS roughly when $k \geq 0.75K$, but underperforms QPS at shorter path lengths by 7.6% (when $k = 21$ and $K = 59$) to 33.5% (when $k = 78$ and $K = 236$), again due to the "inherited" coding inefficiency problem.

RECIPE-t vs RECIPE-d: Figure 2 shows how the (AVST) table size affects the coding efficiency with the path length k varying between 1 and 59. Figure 2 contains three plots representing RECIPE-t with table sizes of 1000, 10,000, and 30,000 rows respectively and one plot representing RECIPE-d (which induces HRS). Figure 2 shows that as the table size becomes larger, the coding efficiencies of RECIPE-t becomes closer to those of RECIPE-d. The zoomed tail section (when k gets close to K) plotted in Figure 2 shows that the difference in coding efficiency between RECIPE-d and RECIPE-t is negligible when the table has 30,000 rows (about 960 KB in size). This is a small cost to pay (for each switch to store this table) in exchange for packets not having to carrying the XOR degree information in them.

VII. Related Work

RECIPE codes are different than the so-called "distributed LT codes" [10], [11] in literature. The latter codes are designed for *multiple-access relay channels*, in which messages come from multiple sources that do not communicate with each other at all. These sources send LT codewords to a common relay node, which further combines (XOR-merges) them to improve the coding efficiency. Our setting is less restrictive in the sense that sources (switches) are allowed to have some limited communication through the TTL field and the XOR degree (explicit in RECIPE-d and implicit in RECIPE-t).

VIII. Conclusion

As the first systemic study of LT codes in the distributed setting, we make the following three contributions. First, we show that not every LT code is feasible (n the distributed setting, and give the sufficient and necessary condition for being RECIPE-feasible. Second, we propose RECIPE, a distributed encoding protocol that can efficiently implement any RECIPE-feasible code. Third, we propose two heuristic search algorithms, namely HRS and QPS, that have led to the discovery of RECIPE-feasible codes that are much more efficient than PINT, the state of the art.

Acknowledgment This material is based upon work supported by the National Science Foundation under Grant No. CNS-1909048 and CNS-2007006.

REFERENCES

[1] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 56–67, 1998, publisher: ACM New York, NY, USA.

[2] M. Luby, "LT codes," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.* IEEE Computer Society, 2002, pp. 271–271.

[3] J. Jiang, M. Mitzenmacher, and J. Thaler, "Parallel peeling algorithms," *ACM Transactions on Parallel Computing (TOPC)*, vol. 3, no. 1, pp. 1–27, 2017, publisher: ACM New York, NY, USA.

[4] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic in-band network telemetry," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 662–680.

[5] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, vol. 15, 2015.

[6] V. Jeyakumar, M. Alizadeh, Y. Geng, C. Kim, and D. Mazières, "Millions of little minions: Using packets for low latency network programming and visibility," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 3–14, 2014, publisher: ACM New York, NY, USA.

[7] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I know what your packet did last hop: Using packet histories to troubleshoot networks," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 71–85.

[8] P. Tammana, R. Agarwal, and M. Lee, "Simplifying datacenter network debugging with {PathDump}," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 233–248.

[9] J. Meng, Z. Liu, Y. Wang, and J. Xu, "Recipe: Rateless erasure codes induced by protocol-based encoding (full version)," 2023. [Online]. Available: https://arxiv.org/abs/2305.03795

[10] S. Puducheri, J. Kliewer, and T. E. Fuja, "Distributed lt codes," in *2006 IEEE International Symposium on Information Theory*, 2006, pp. 987–991.

[11] ——, "The design and performance of distributed lt codes," *IEEE Transactions on Information Theory*, vol. 53, no. 10, pp. 3740–3754, 2007.