Methods for Efficient Network Coding

Petar Maymounkov and Nicholas J. A. Harvey
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
E-mail: {petar,nickh}@csail.mit.edu

Desmond S. Lun
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
E-mail: dslun@mit.edu

I. Introduction

Random linear network coding is a multicast communication scheme in which all participating nodes send out coded packets formed from random linear combinations of packets received so far. This scheme is capacity-achieving for single sessions over lossy wireline or wireless packet networks [1], [2], [3], [4]. Thus, from the point of view of efficiently utilizing transmissions, random linear network coding is an attractive strategy. However, it is not presently attractive from the point of view of efficiently utilizing *computational* resources.

To decode a k-packet-long message, a decoder needs to invert a $k \times k$ dense matrix, which, using Gaussian elimination, requires $O(k^3)$ operations (or $O(k^2)$ operations per input symbol). Once the matrix inverse is computed, applying the inverse to the received coded packets to recover the message requires $O(k^2)$ operations (or $O(k)$ operations per input symbol). Although the former computation is more costly than the latter as a function of k, it is usually the latter cost that dominates, since the latter computation depends on the length of the packets (which is usually on the order of kilobytes), while the former does not. This dominant cost can make the computational resources required for random linear network coding prohibitive [5].

But a random linear code is a somewhat naïve code. By picking the code randomly, we ensure that the code is efficient at communicating information but, at the same time, by employing little design in the code, we obtain a code that is computationally inefficient. Thus, a natural code design question arises: *can we design a network code that preserves the communication efficiency of a random linear code and achieves better computational efficiency?*

In this paper, we give an affirmative answer to this question. The code that we present as our answer achieves significantly better computational efficiency and is based primarily on techniques that are now quite standard in the literature on erasure codes. That said, our application of these techniques to network coding requires some novel ideas. First, we partition the messages into "chunks" (smaller sub-messages of the original message) and introduce a simple scheme for deciding which chunks to transmit at any point in time. (The use of chunks allows us to achieve sub-quadratic computational costs.) Second, we introduce a precode, which allows the message to be recovered when only a subset of the full set of chunks have been successfully decoded, thereby achieving linear computational costs. For simplicity the exposition in this paper focuses on the unicast communication case (with a single source and a single sink), however it should be readily clear that our arguments immediately extend to the multicast communication case (with a single source and multiple sinks).

This work was supported by National Science Foundation under grant nos. CCR-0093349, CCR-0325496 and CCF-0515221; by the Army Research Office through University of California subaward no. S0176938; by the Office of Naval Research under grant nos. N00014-05-1-0148 and N00014-05-1-0197; and by the Natural Sciences and Engineering Research Council of Canada. This research was also conducted as part of the IRIS project (`http://project-iris.net/`), supported by the National Science Foundation under cooperative agreement no. ANI-0225660.

A. Comparison to Related Work

Chunking schemes have been previously applied in the context of network coding. For example, Chou et al. [6] divide the message into "generations" and code each generation separately using random linear network codes. However, their use of generations was not motivated by computational efficiency, but rather as a means of handling the lack of synchronization in practical networks. An important issue that one must address in any chunking scheme is: when a node generates a symbol for transmission, which chunk's data should be used to generate the symbol? Chou et al. propose two designs addressing this issue. In the first design, receiver(s) provide feedback to the sender about which generations have already been completely decoded. The sender can therefore stop transmitting symbols for completed generations and thus nodes transmit symbols for generations in a sequential order. In their second design, nodes simply transmit symbols for generations in a round-robin order.

Our design address the aforementioned issue differently: *each node chooses a chunk at random and produces a coding symbol for that chunk*. Our design has several benefits: first, feedback from receivers is not required; and second, good coding performance is ensured under *any* pattern of erasures by the channel (even an adversarially chosen pattern). The round-robin design does not have this latter property, since erasures could exhibit some regular pattern. Two drawbacks of our design are that the internal network nodes require storage proportional to the size of the original message, and that streaming media is not supported. However, our design is very applicable to the common scenario of peer-to-peer file distribution [7], [8].

Erasure code	Overhead	Encoding cost	Decoding cost
LT	$O(\sqrt{k}\ln^2 k)$	$O(\ln k)$	$O(\ln k)$
Raptor	λk	$O(\ln 1/\lambda)$	$O(\ln 1/\lambda)$
Network code	**Overhead**	**Encoding cost**	**Decoding cost**
Chunked w/o precode	$O(k/\ln^{1/4} k)$	$O(\ln^2 k)$	$O(\ln^2 k)$
Chunked w/ precode	λk	$O(\lambda^{-4}\ln 1/\lambda)$	$O(\lambda^{-4}\ln 1/\lambda)$
Dense	$O(\log k)$	$O(k)$	$O(k)$

TABLE I

COMPARISON OF VARIOUS ERASURE CODES AND NETWORK CODES WITH ERROR PROBABILITY $\mathrm{poly}(1/k)$.

Precodes have been intensively studied in the context of efficient erasure codes, but to the best of our knowledge, they have not been used in the context of network codes. Precodes originate from work on Raptor codes [9] and Online codes [10], which we collectively refer to as Fountain codes. These codes are designed to operate over a single erasure channel and therefore do not function well in a general network (they incur loss of rate, additional delay, or both). That said, one would expect Fountain codes to be more efficient than network codes on a single erasure channel because this scenario is a very restricted case of a lossy network. These differences notwithstanding, it is instructive to compare our codes (Chunked codes with and without a precode) to efficient erasure codes (Fountain codes and LT codes [11], which are Fountain codes without a precode) and random linear network codes, which we call Dense codes.

In Table I, we show the overhead, encoding cost, and decoding cost necessary to achieve error probability that varies at most inversely polynomial in k (i.e., a probability of unsuccessful decoding of at most $1/k^c$ for some positive c) for these various codes. Overhead, in the context of an erasure code is defined as the number of received symbols, n, minus k; in the context of a network code, the definition is similar, but more complicated—we defer the precise definition of the overhead of a network code to Section II. Encoding cost is defined as the number of packet operations performed by a coding node divided by k. Decoding cost is defined as the number of packet operations performed by a decoding node divided by k (this cost excludes, for example, the cost of calculating an inverse, since these are bit or field operations rather than packet operations). We see that Chunked codes, whether with or without a precode, achieve decreased encoding and decoding costs compared to Dense codes. These improved encoding and decoding costs come at the cost of increased overhead, as one would expect. Moreover, we see that Chunked codes achieve similar performance to LT codes and Raptor codes, though, as expected, the erasure codes are more efficient. Note that, for codes with a precode (Raptor codes and Chunked codes with a precode), there is a parameter $\lambda > 0$ that allows for a range of performances to be achieved.

B. Network Model

We conduct our analysis of Chunked codes and Dense codes under the network model presented in Section II. This model is based on the concept of a *time-expanded graph* or *trellis* [3], [8]. We use instead the term *adversarial schedule*. An adversarial schedule is a deterministic, graph-theoretic description of a specific transmission instance, whereby only the successfully transmitted packets are specified. Such an instance may be chosen by an adversary, hence the term. Our adversarial model is stronger than the probabilistic models used in [1], [2], [3], [4]. As in previous work, our model allows for a convenient graph-theoretic upper-bound on the capacity that any coding scheme can achieve.

The Chunked and Dense coding schemes both asymptotically achieve this upper-bound. Moreover, both schemes are intrinsically *rateless* (a.k.a., adaptive), and allow network nodes to construct and transmit new coded symbols before they have completely received and decoded the entire k-packet message. We believe it is interesting that the Chunked codes can achieve these useful properties while also greatly improving the computational efficiency of the Dense codes.

The physical networks considered in this paper are wireline networks, not wireless; this restriction is for simplicity and clarity of exposition. We expect that our network model and results can be extended to wireless networks. Thus, we expect that, in wireless networks, Chunked codes will solve the problem of determining what data to place in packets in a rate-optimal way for a single session (as Dense codes do [4]). Scheduling, routing, and subgraph selection (see, for example, [12]) are important problems that are outside the scope of the coding problem we consider.

C. Organization

Under the adversarial network model, we conduct an analysis of Dense codes in Section III, then an analysis of Chunked codes in Section IV. Our analysis of Dense codes adds to the existing knowledge of such codes developed in [1], [2], [3], [4]. In particular, we believe that our analysis is the first to quantify the overhead associated with a Dense network code. Moreover, this analysis of Dense codes is necessary for our analysis of Chunked codes. In short, Chunked codes operate by dividing the message into chunks (sub-messages), each of which is sent by a Dense code. At each time step, each node randomly chooses a chunk to operate on. The number of chunks to use must be selected with care, since too few chunks implies little gain in encoding or decoding cost, and too many chunks leads to "the curse of the Coupon Collector", where a decoder may need to wait a long time for the last few chunks it needs. Chunked coding is described in more detail

in Section IV. In Section V, we describe how precoding may be used with a Chunked code to achieve linear computational costs (in the input message size).

II. Formal Network Model

Let V be the set of nodes participating in a packet network, with $s \in V$ a distinguished source and $t \in V$ a distinguished sink. Unlike typical formalizations, we will not explicitly model network channels as edges. Instead, our model will describe packets which were successfully transmitted between two network nodes. A packet that was transmitted from node v_1 at time t_1 and received by node v_2 at time t_2 is modeled as the 4-tuple (v_1, t_1, v_2, t_2). We do not assume that transmissions are generated according to any probabilistic process. Instead, we assume that they may be chosen by an "adversary" which seeks to minimize the rate achieved by the coding scheme under consideration. Therefore, a set Ψ of transmissions that occur in a particular network scenario is called an *adversarial schedule*.

Transmissions have a natural graphical interpretation. Define a graph $G_T(\Psi)$ whose vertex set is a subset of $V \times (\mathbb{R} \cup \{\infty\})$. Each transmission induces two end-vertices and a single edge (called a *traffic edge*) in the graph. There is no loss of generality in assuming that traffic edges have unit capacity. To complete the description of $G_T(\Psi)$, we add an additional source vertex $(s, 0)$ and sink vertex (t, ∞). Also, for any two vertices (w, t) and (w, t') with $t < t'$, we add an infinite-capacity edge between these vertices. The graph $G_T(\Psi)$ is called a *time-diagram*, or *continuous-time trellis*.

There is a natural upper bound on the number of distinct packets that can be transmitted from s to t under an adversarial schedule Ψ. This bound is given by the capacity of the minimum cut separating $(s, 0)$ and (t, ∞) in $G_T(\Psi)$. This quantity is called the *capacity* of the schedule. The capacity of a schedule Ψ represents the maximum number of packets that we can hope to communicate across it. Therefore, if a network code communicates k packets over a schedule of capacity n, then its *overhead* is $n - k$. If the schedule Ψ were known in advance, it would be trivial to construct a coding scheme with zero overhead. The issue is that Ψ is of course not known in advance, and therefore one wishes to achieve low overhead via an efficient coding scheme.

Given an adversarial schedule, construct a maximum cardinality collection of paths between $(s, 0)$ and (t, ∞). By the max-flow min-cut theorem, the number of such paths is precisely the capacity n. These paths are grouped into equivalence classes based on whether they traverse the same sequence of network nodes. An equivalence class of paths is called a *flow path*. For a flow path ξ, its *capacity*, denoted $c(\xi)$, is the number of paths in G_T that are associated with ξ. Thus a flow path ξ can be viewed as transmitting $c(\xi)$ packets from s to t. Let $\mathcal{F}$ be our collection of flow paths, and note that $\sum_{\xi \in \mathcal{F}} c(\xi)$ equals the capacity n of Ψ.

III. Dense Code Analysis

In this section we give a tight analysis of the performance of random linear codes under adversarial schedules. Here, we assume that each packet transmitted by a node is a random linear combination of *all* previously received packets, i.e., the linear combination is *dense*. The results in this section will be used in the following section to obtain more computationally efficient codes. The analysis proceeds as follows.

A. Preliminaries

The source node is given an input message consisting of k symbols. The symbols are drawn from a vector space over $\mathbb{F}_2$, i.e., symbols are strings of bits. Each symbol transmitted by any node in the network will be some linear combination of the input message symbols. The coefficients of this linear combination are called the *payload vector* associated with the transmission. Under our coding scheme, the payload vectors are random variables.

Suppose that we are given a schedule Ψ with capacity n. First, we construct a collection $\mathcal{F}$ of flow paths, as described above. Consider a particular flow path ξ with capacity $c(\xi)$. Suppose that the nodes on ξ are $s = v_0, v_1, \ldots, v_{\ell(\xi)} = t$, where $\ell(\xi)$ denotes the length of ξ. For $0 \leq i < \ell(\xi)$, define a matrix Q_i whose columns are the $c(\xi)$ payload vectors corresponding to the transmissions (on flow paths) between vertex v_i and v_{i+1}. So the size of Q_i is $k \times c(\xi)$.

Our subsequent analysis measures the amount of information transmitted on an edge of ξ by the "randomness" of the corresponding matrix Q_i. The following definitions formalize this idea. Let M be an $r \times n$ matrix whose entries are each Bernoulli random variables.

Definition 3.1: A set of columns of M are called *dense* if their entries are uniform and i.i.d. The *density* of M, denoted $\mathcal{D}(M)$, is the maximum value d such that M contains d dense columns. If $\mathcal{D}(M) = n$ then we say that the matrix M is dense.

Definition 3.2: For any $\alpha \in \mathbb{N}$, we say that M contains a *α-triangle* if there is a permutation π on the rows such that from row $\pi(i)$ we may choose at least $\alpha - i + 1$ entries that are uniform, and such that all chosen variables are mutually independent. The *triangular-density* of M, denoted $\mathcal{T}(M)$, is defined to be the maximum value of α such that M contains a α-triangle.

Clearly $0 \leq \mathcal{D}(M) \leq \mathcal{T}(M) \leq n$. Our definitions of density are motivated by the following lemma. The key point of the lemma is that the probability of low rank is exponentially decreasing in both $(\alpha - r)$ and γ.

Lemma 3.3: Let M be as above, and suppose that $\mathcal{T}(M) = \alpha$, where $\alpha \geq r$. Then, for any $\gamma \in \mathbb{N}$, we have $\Pr[\,\mathrm{rank}(M) < r - \gamma\,] \leq (r - \gamma) 2^{-(\alpha - r + \gamma + 1)}$.

Proof. By assumption, M contains an α-triangle; let π be the corresponding permutation. Let M' be M restricted to the first $r - \gamma$ rows under π. Clearly $\mathcal{T}(M') \geq \alpha$. Consider the event that M' does not have full row-rank. This is precisely the event that there exists a 0-1 vector u of length $r - \gamma$ with $u^{\mathsf{T}} M' = 0$. Consider a vector u whose first entry with value 1

is the i^{th} entry (all previous entries are 0). There are $2^{r-\gamma-i}$ such vectors. By the definition of an α-triangle, at least $\alpha-i+1$ of the values in row i are uniform and i.i.d. Therefore at least $\alpha - i + 1$ entries of the vector $u^{\mathsf{T}}M'$ are uniform and i.i.d. All of these entries are 0 with probability at most $2^{-(\alpha-i+1)}$. Taking a union bound over all vectors u, the probability that $u^{\mathsf{T}}M'$ is 0 for any vector u is at most $(r-\gamma)2^{-(\alpha-r+\gamma+1)}$. ■

The following corollary (while not a part of our formal arguments) intuitively justifies our definition of density as a measure of information.

Corollary 3.4: Let M be as above and assume that $\mathcal{T}(M) = \alpha$, where $\alpha \geq r$. Let x be a 0-1 vector with uniform and i.i.d. entries. Then, for any $\gamma \in \mathbb{N}$, we have $H(Mx \mid M) \geq (r-\gamma) \cdot \big(1 - (r-\gamma) \cdot 2^{-(\alpha-r+\gamma+1)}\big)$.

Lemma 3.3 is similar in spirit to the following well-known result. (See, e.g., Shokrollahi [9, Proposition 2].)

Lemma 3.5: Let M be as above, and assume that $\mathcal{D}(M) = n \geq r$. Then M fails to have full row-rank with probability at most $2^{-(n-r)}$.

The following lemma is also useful. Its proof is straightforward and hence omitted.

Lemma 3.6: Let M be as above and suppose that M is dense, i.e., $\mathcal{D}(M) = n$. Let R be a matrix with n rows and $\operatorname{rank}(R) \geq s$. Then $\mathcal{D}(MR) \geq s$.

B. Proof Outline

We wish to analyze the overhead of Dense coding, i.e., the number of packets transmitted from s to t that suffice to deliver the message. Our approach is to consider each flow path ξ in isolation[1], and analyze the amount of information transmitted along the path. To do so, we consider the sequence of matrices $Q_1, \ldots, Q_{\ell(\xi)}$ and analyze their density. The definition of our coding scheme immediately implies that $\mathcal{D}(Q_1) = c(\xi)$. Next, we argue that the density loss at each vertex $\mathcal{D}(Q_i) - \mathcal{D}(Q_{i+1})$ is small, and hence $\mathcal{D}(Q_{\ell(\xi)-1})$ is large. If $\mathcal{D}(Q_{\ell(\xi)-1})$ is just slightly larger than $c(\xi)$ for all paths ξ (and hence $\sum_{\xi\in\mathcal{F}} \mathcal{D}(Q_{l(\xi)-1})$ is slightly larger than the input message size), then Lemma 3.5 shows that with good probability the sink can decode the message by solving a system of equations.

C. Density loss is small

First we consider the loss of density at a particular node.

Lemma 3.7: For any $i \geq 0$ and $\epsilon > 0$, the inequality $\mathcal{D}(Q_i) - \mathcal{D}(Q_{i+1}) \leq \log \mathcal{D}(Q_i) + \log 1/\epsilon$ fails to hold with probability at most ϵ.

Proof. Recall that each column of Q_i is a payload vector corresponding to a transmission between v_i and v_{i+1}. Let $d = \mathcal{D}(Q_i)$, and let Q_i' be a submatrix of Q_i consisting of d dense columns.

The proof relies on two observations. (1) In our coding scheme, the j^{th} transmission between v_{i+1} and v_{i+2} is a random linear combination of the transmissions that were already received at v_{i+1}. (2) By the time that this transmission is sent, node v_{i+1} will have received at least j transmissions from v_i, plus possibly some additional unrelated transmissions. (This follows from our construction of ξ.) Let us interpret these observations in terms of matrices.

Observation (1) implies that each column of Q_{i+1} is a random linear combination of columns of Q_i, plus possibly some (independent combinations of) additional vectors. We will only concern ourself with linear combinations of columns in Q_i', so we may write $Q_{i+1} = Q_i' \cdot R + N$, for some matrix N. Each entry of R and N is either zero, or contains a random variable. These random variables are all independent, implying that $\mathcal{D}(Q_{i+1}) \geq \mathcal{D}(Q_i' \cdot R)$.

Observation (2) implies that $\mathcal{T}(R) = d$. So Lemma 3.3 implies that $\operatorname{rank}(R) \geq d - \gamma$ fails to hold with probability at most $(d-\gamma)2^{-\gamma-1}$. When this inequality holds, Lemma 3.6 shows that $d - \gamma \leq \mathcal{D}(Q_i' \cdot R) \leq \mathcal{D}(Q_{i+1})$. Setting $\gamma = \log d + \log(1/\epsilon)$ concludes the proof. ■

The preceding lemma immediately yields a bound on the cumulative loss of density.

Lemma 3.8: For any $j \geq 0$ and $\epsilon > 0$, the inequality $c(\xi) - \mathcal{D}(Q_j) \leq j \cdot (\log c(\xi) + \log 1/\epsilon + \log j)$ fails to hold with probability at most ϵ.

Proof. Applying Lemma 3.7 and observing that $\mathcal{D}(Q_i) \leq c(\xi)$ shows that the inequality

$$\mathcal{D}(Q_i) - \mathcal{D}(Q_{i+1}) > \log c(\xi) + \log 1/\epsilon + \log j$$

holds with probability at most ϵ/j. Recalling that $\mathcal{D}(Q_1) = c(\xi)$, we have

$$\mathcal{D}(Q_j) \;=\; c(\xi) - \sum_{i=1}^{j-1} \big(\mathcal{D}(Q_i) - \mathcal{D}(Q_{i+1})\big).$$

Applying a union bound over all $1 \leq i < j$ shows that the inequality

$$\mathcal{D}(Q_j) \;<\; c(\xi) - j \cdot \big(\log c(\xi) + \log 1/\epsilon + \log j\big)$$

holds with probability at most ϵ. ■

Now, we consider the loss of density over all paths ξ. Suppose that $\mathcal{F}$ consist of p flow paths, and let $\ell_{\max} = \max_\xi \ell(\xi)$ be the length of the longest one. Let $n = \sum_\xi c(\xi)$ denote the total capacity of the schedule. Also, let $Q^{(\xi)}$ denote the final matrix (of payload vectors) on path ξ. Then Lemma 3.8 together with another union bound shows that the inequality

$$n - \sum_\xi \mathcal{D}(Q^{(\xi)}) \;\geq\; p\ell_{\max}\big(\log n + \log 1/\epsilon + \log(p\ell_{\max})\big) \quad (1)$$

holds with probability at most ϵ.

D. Decoding at the sink

The output symbols available at the sink are then described by the matrix $L = [Q^{(\xi_1)} | \ldots | Q^{(\xi_p)}]$. (Here the "|" symbol denotes column-wise matrix concatenation.) Clearly $\mathcal{D}(L) = \sum_\xi \mathcal{D}(\xi)$. Successful decoding occurs when L has rank k. Our discussion of (1), together with Lemma 3.5 yields our main theorem for dense codes and adversarial schedules.

[1]We remark that analyzing the flow paths separately does not in general yield an optimal analysis. Moreover, an adversarial schedule Ψ might decompose into flow paths $\mathcal{F}$ in several ways, and different $\mathcal{F}$'s may yield better or worse analyses.

Theorem 3.9: For any adversarial schedule with capacity at least

$$k + pl_{\max}\big(\log k + \log 1/\epsilon + \log(pl_{\max}) + 1\big) + \log 1/\epsilon + 1,$$

our coding scheme fails to deliver an input message of size k symbols to the sink with probability at most ϵ.

In particular, if $pl_{\max} \log 1/\epsilon = o(k)$ and $\epsilon = o(1)$, the scheme has vanishing overhead and vanishing error probability. For instance, we may take $\epsilon = e^{-\sqrt{k}}$ and $pl_{\max} = o(\sqrt{k}/\log k)$.

IV. Chunked Code Analysis

In this section, we will view the input message as being logically partitioned into q contiguous chunks of $\alpha = k/q$ symbols. We will refer to α as the *aperture*[2] of the code. The coding scheme at each node in the network is as follows: the node chooses a chunk at random, and transmits a symbol according to the dense coding scheme (applied to that chunk only).

Our analysis of the chunked coding scheme proceeds in two parts. For any fixed adversarial schedule, we first show that the schedule's flow is distributed evenly among all chunks with high probability (Section IV-A). Then we apply the dense coding result (as in Section III) to argue that all chunks decode (Section IV-B).

A. Flow for a Fixed Chunk ω

In this section we will consider a fixed flow path $\xi = (v_0, v_1, \ldots, v_{\ell(\xi)})$ and we will show that all chunks receive a roughly even allocation of the flow. First, some terminology is needed. Let $n = c(\xi)$, and let $l = \ell(\xi)$. If $\omega \in [q]$ is a particular chunk, and a transmission contains information pertaining to chunk ω, then we call it an *ω-transmission*.

Theorem 4.1: For each chunk $\omega \in [q]$, the ω-transmissions form a flow of capacity at least $\big(1 - \tilde{O}\left(l^{3/4}(q/n)^{1/4}\right)\big) \cdot (n/q)$ with probability no less than $1 - \epsilon$, for $\epsilon > 0$, so long as

$$l^3 \cdot \ln(l/\epsilon) \;=\; o(n/(q \log n)).$$

Thus, in order to have a capacity-achieving code, the length of each flow path should not be too long, and the capacity of the flow path should be large (even when divided into q chunks). In practice, one might imagine using our coding scheme to send large files on the internet. Each symbol might have size 1KB, a file might contain tens or hundreds of thousands of symbols, and communication paths might have 30 hops. In this scenario, the assumption of Theorem 4.1 would certainly be satisfied, so long as the number of chunks is roughly one thousand.

Proof Sketch: The statement of Theorem 4.1 seems quite intuitive. By standard concentration arguments, any edge in the flow path has roughly a $1/q$ fraction of its transmissions chosen to be ω-transmissions. However, this fact alone does not yield the theorem. One complicating factor is that ω-transmissions must be evenly "spread out" in order to ensure that they lead to valid flow paths. For example, if node v_i chose its last n/q transmissions as ω-transmissions, and node v_{i+1} chose its first n/q transmissions as ω-transmissions, then the chunk ω could have no flow paths whatsoever. However, such an arrangement of ω-transmissions is unlikely, and this observation is what we formalize below.

Fix a particular node v_i. We will consider three categories of transmissions: those arriving at v_i, those departing from v_i, and those arriving at v_{i+1}. The latter two categories of transmissions are identical content-wise, but they may have been reordered by the channel. This issue makes it convenient to treat the two categories separately.

Our analysis partitions each category of transmissions into b contiguous "buckets", where b is a parameter to be chosen later. The first n/b transmissions are in the first bucket, the next n/b transmissions are in the second bucket, and so on. For each bucket, a standard concentration argument shows that the number of ω-transmissions will be not much less than its expectation with high probability. So we may fix a value μ' which is slightly less than the expectation, and choose precisely μ' ω-transmissions from each bucket. These chosen transmissions are called *good*, and the process for choosing good transmissions will be specified shortly.

Now to show that the ω-transmissions have many flow paths, we want each ω-transmission departing node v_i to be "matched up" with a ω-transmission that arrived earlier at node v_i. Our bucketing scheme[3] makes this straightforward. The flow paths will consist of good transmissions. For $j < b$, all good ω-transmissions in the j^{th} bucket arriving at v_i can be matched with good ω-transmissions in the $(j+1)^{\text{th}}$ bucket departing v_i. Any ω-transmission in the b^{th} bucket arriving at v_i is considered to be lost.

There is one remaining issue to be addressed. Each transmission from node v_i to node v_{i+1} belongs to two categories: departing transmissions of node v_i and arriving transmissions of node v_{i+1}. A transmission may be considered "good" in either or both of those categories. However, the transmission can only be used on a flow path if it is good from the perspective of *both* nodes, so we must exercise some care in choosing transmissions to be good. Let us say that a transmission is *half-good* if it is good from the perspective of one node, but not the other. Half-good transmissions are undesirable because they prevent formation of a flow path.

The probabilistic method [14] allows us to choose good transmissions in a particularly simple way. As mentioned above, we may assume that the number of ω-transmissions in each bucket is close to its expectation. Consider choosing μ' of these transmissions uniformly at random to be good. By choice of μ', an ω-transmission is good from the perspective of node v_i with probability close to 1, say $1-\rho$. An identical argument holds for node v_{i+1}, so an ω-transmission is good from the

[2]This concept is derived from an ongoing project [13] which seeks to improve the cache-performance of rateless erasure codes. Section VI-C discusses this work further.

[3]It has come to our attention that Wu [3] has independently developed a similar analysis.

perspective of *both* nodes with probability at least $1-2\rho$. Thus, in expectation, the number of half-good ω-transmissions is small. Therefore, there exists a choice of good ω-transmissions that does not cause many half-good ω-transmissions.

To conclude the argument, any flow path of ω-transmissions which arrived at node v_i but could not reach node v_{i+1} can either be attributed to landing in the b^{th} bucket at node v_i, or to a half-good transmission. The number of such unfortunate occurrences is small, and therefore the majority of the flow paths can indeed continue onwards to node v_{i+1}.

Proof Details: Consider our bucketing scheme described above. The expected number of ω-transmissions in a given bucket is $\mu := n/bq$. There exists a value μ' such that this random variable is at least μ' with probability at least $1-\epsilon$. A Chernoff bound [14] shows that we may take $\mu' := \mu - c'\sqrt{\mu}$, where c' is a function of ϵ. Similarly, the random variable is at most $\mu'' := \mu + c''\sqrt{\mu}$ with probability at least $1-\epsilon$. We will specify c' and c'' later.

The probability that a ω-transmission is not chosen to be good in a given category is at most

$$\begin{aligned}\rho \;&:=\; (\mu'' - \mu')/\mu' \;=\; (c''+c')\sqrt{\mu}/(\mu - c'\sqrt{\mu}) \\ &=\; O\big((c''+c')/\sqrt{\mu}\big).\end{aligned}$$

By the Chernoff bound mentioned above, the number of ω-transmissions is $\Theta(b\mu)$ with high probability. Our probabilistic construction ensures that the number of half-good transmissions is at most a 2ρ fraction of this quantity, i.e.,

$$2\rho \cdot \Theta(b\mu) = O((c''+c')b\sqrt{\mu}) = O((c''+c')\sqrt{nb/q}).$$

On the other hand, the number of ω-transmissions lost due to landing in the b^{th} bucket is at most μ''. Summing over all vertices in the flow ξ, the total number of lost ω-transmissions is

$$O\Big(l\big(n(c''+c')/\sqrt{\mu} + \mu''\big)\Big) \tag{2}$$

$$=\; O\Big(l\big((c''+c')\sqrt{nb/q} + (n/bq)\big)\Big). \tag{3}$$

We now specify c' and c'' such that the bound of (2) holds with good probability. We desire that the ω-transmissions are well distributed in each bucket at each node, and for each $\omega \in [q]$. Therefore we must union bound over b, l, and q. A standard application of Chernoff bounds shows that we achieve an overall failure probability of ϵ by taking both c' and c'' to be $O\big(\sqrt{\log(lbq/\epsilon)}\big) = O\big(\sqrt{\log(ln/\epsilon)}\big)$.

The final parameter remaining to be specified is b; we address this issue now. We define $b = \left\lceil \sqrt{n/ql(c''+c')} \right\rceil$; the ceiling has negligible effect so long as $ql(c''+c') = o(n)$. The number of lost ω-transmissions now becomes

$$\begin{aligned}&O\Big(l\big((c''+c')\sqrt{nb/q} + (n/bq)\big)\Big) \\ &=\; O\left(l\sqrt{\frac{n(c''+c')}{q}}\left(\left(\frac{n}{ql}\right)^{1/4} + \sqrt{l}\right)\right).\end{aligned}$$

As argued above, the total number of ω-transmissions is $\Theta(b\mu) = \Theta(n/q)$ with high probability. Therefore a simple calculation shows that the number of lost ω-transmissions is asymptotically less than the total number, so long as $l^3 \cdot (c'+c'')^2 = o(n/q)$, i.e., $l^3 \cdot \log(ln/\epsilon) = o(n/q)$. This completes the proof of Theorem 4.1.

B. All Chunks Decode

The main result of this paper is the performance of chunked coding on adversarial schedules:

Theorem 4.2: Let an arbitrary adversarial schedule with maximum flow path length $l_{\max}$, and decomposition into p flow paths be given, and let $n = \sum_\xi c(\xi)$. An input message of k symbols can be delivered from the source to the sink with probability of failure no bigger than $\epsilon > 0$, as long as:

- $l_{\max}^3 \cdot \ln(k\,p\,l_{\max}/\epsilon) = o(k/q)$, and
- $n = (1+o(1)) \cdot \Big(k + p\,l_{\max}\,q\log(k\,p\,l_{\max}/\epsilon) + q\log q + q\log 1/\epsilon + q\Big)$

Furthermore, the encoding, decoding, and recoding algorithms take k/q symbol operations per input message symbol.

This theorem follows immediately from Theorem 4.1, combined with a union bound over all chunks, and an application of Theorem 3.9. In one particular instance of arguments choice, we obtain that:

Corollary 4.3: Under the conditions of Proposition 4.2 where additionally $l_{\max}p = O(1)$ and $q = k/\log^2 k$, chunked coding requires $\log^2 k$ symbol operations per input symbol for encoding, decoding and recoding, and achieves probability of error $\epsilon = 1/k^c$ for arbitrary constants $c > 0$. Furthermore, the reception overhead $h(k)$ satisfies $h(k) = O(k/\log^{1/4} k) = o(k)$. (The derivation of this corollary is deferred to the Appendix.)

This corollary allows us to contrast chunked coding for adversarial schedules to LT coding for regular erasure channels. In both cases the error probability decreases as $O(1/k^c)$. Chunked codes are computationally more expensive, requiring $O(\log^2 k)$ operations per input symbols, compared to $O(\log k)$ for LT codes. And finally, the overhead of chunked codes decreases as $O(k/\log^{1/4} k)$, compared to $O(\sqrt{k}\log^2 k)$ for LT codes.

V. Linear-time Codes

In [15] Luby et al. introduced the concept (and the first instance) of a *practical* erasure code, which was later improved to a *practical rateless* erasure code in [9], [10]. A practical rateless code is one whose encoding algorithm produces an output symbol in $O(1)$ time (we use the term *time* to refer to symbol operations) after a linear-time, i.e., $O(k)$, preprocessing stage, and whose decoding algorithm recovers the input message in linear time (or equivalently $O(1)$ time per input symbol). Additionally, a practical code is parameterized by its overhead $\lambda > 0$ and probability of failure $\epsilon > 0$: for a message of size k, the decoder fails to recover the original message after receiving $n = (1+\lambda)k$ output symbols with probability at most ϵ. Practical codes are distinguished from classical Shannon codes (and from LT codes) in two ways. (1) They have an arbitrarily small, but fixed, overhead that

Code	Preprocess Cost	Encoding cost	Decoding cost
Fountain	$O(k \ln 1/\lambda)$	$O(\ln 1/\lambda)$	$O(\ln 1/\lambda)$
Chunked	$O(k \ln 1/\lambda)$	$O(\lambda^{-4} \ln 1/\lambda)$	$O(\lambda^{-4} \ln 1/\lambda)$

TABLE II
COMPARISON OF PRACTICAL CODES WITH ERROR PROBABILITY $\mathrm{poly}(1/k)$.

does not vanish with the input size. (2) Their probability of error vanishes (slowly) as $\mathrm{poly}(1/k)$; in other words, practical codes have a zero-exponent probability of error. In the case of Tornado codes, the error probability is a non-vanishing but arbitrarily small constant.

Naturally, the magnitude of λ and ϵ influences the running time of the code (independently of k). The Raptor and Online [9], [10] code designs (referred to as Fountain Codes) with appropriate choice of precode achieve poly-logarithmic dependence on λ when $\epsilon = \mathrm{poly}(1/k)$. For comparison with Fountain Codes, we developed a practical version of Chunked Codes which combines the original design with a suitable precode. Table II summarizes our results, which follow from:

Theorem 5.1: For any given overhead $\lambda > 0$ and any network with $\varphi = p \cdot l_{\max}$, there is a Practical Chunked Code that preprocesses the input message in $O(k \ln 1/\lambda)$ symbol operations, computes an output symbol in $O\big((\varphi^3/\lambda^4)\ln(\varphi/\lambda)\big)$ symbol operations at the source as well as at intermediate nodes, decodes the message in $O\big((\varphi^3/\lambda^4)\ln(\varphi/\lambda)\big)$ symbol operations per input symbol, and fails to decode with probability no larger than $1/k^c$ for any constant c.

We will also remark, without proof, that when the network is reduced to a single link, i.e. the classical Erasure Channel setting, the above theorem still holds while the $O\big((\varphi^3/\lambda^4)\ln(\varphi/\lambda)\big)$ terms are replaced by $O\big(\lambda^{-2}\ln(1/\lambda)\big)$.

A. Proof Sketch

The idea of the construction is similar to the one used in the Raptor Codes design. First, a precode is applied to the k input symbols to produce roughly $n \approx (1+\lambda/2)k$ intermediate symbols. The precode does not have to be capacity achieving, but it has to be able to recover the k input symbols from $(1+\lambda/4)k$ intermediate symbols in linear-time with error probability no larger than $1/k^c$. Next we apply a slightly-modified variant of the Chunked Code to the intermediate symbols. This modified code can recover the necessary $(1+\lambda/4)k$ intermediate symbols from $(1+\lambda)k$ output symbols in linear time with exponentially vanishing probability of error.

Observe that the use of the precode eliminates the requirement that the Chunked Code achieves capacity, thus evading "the curse of the Coupon Collector". In turn, this allows us to shrink the aperture (number of symbols per chunk) to a constant size, thereby reducing all computational costs to linear.

B. Proof Details

Precode: Our formal requirements for the precode are: it can be any erasure correcting code of dimension k, block size n, rate $R = (1+\lambda/2)/(1+\lambda)$, and it must be able to recover successfully from a $(\lambda/4)/(1+\lambda) = (1-R)/2$ fraction of erasures with error probability $1/k^c$. (This setup is identical to the one used in [9].) A variety of codes fulfill these requirements. One particular choice is a right-regular LDPC code with message edge degree distribution $\Omega(x) = (2x+3x^2)/5$ (i.e., 2/5th of the edges have message-side degree 2, and 3/5th have degree 3). The properties of these codes are discussed in detail in [16].

Linear-time Chunked Code: To show that a Chunked Code can be concatenated with a precode, we will prove:

Theorem 5.2: For any $\lambda_* > 0$, $\theta_* > 0$ and a network with $\varphi = p \cdot l_{\max}$, the Chunked Code with aperture

$$\alpha = O\left(\frac{\varphi^3}{\lambda_*^4}\log\frac{\varphi}{\theta_*\lambda_*}\right)$$

decodes a $(1-\theta_*)$-fraction of the n intermediate message symbols from $(1+\lambda_*)n$ output symbols with error probability $e^{-O(n)}$.

Proof. As in Section IV-A we focus on a fixed chunk ω. Let ρ be the probability that ω fails to decode. We will show that, for α as above, we have $\rho < 2\theta/3$. Hence the expected number of chunks that fail to decode is at most ρn. Moreover, the number of failed chunks is tightly concentrated around this expectation. This concentration follows from a standard martingale argument where the i^{th} in the martingale sequence reveals which chunk is related to the i^{th} output symbol. (See, e.g., Alon and Spencer [14].) Therefore the theorem follows so long as we can analyze ρ.

For any integer flow value $f > 0$, we have that $\rho \le \rho_R + \rho_F$ where ρ_F is the probability our code's random choices fail to provide chunk ω with a flow of capacity f, and ρ_R is the probability that ω fails to recover conditioned on having received a flow of capacity at least f. We aim to show that:

$$\rho_F < \theta_*/3, \text{ and} \tag{4}$$

$$\rho_R < \theta_*/3 \tag{5}$$

By Theorem 3.9 inequality (5) holds as long as:

$$f = \alpha + \varphi\left(\log\alpha + \log 3/\theta_* + \log\varphi + 1\right) + \log 3/\theta_* + 1$$

In order to take care of inequality (4) we need to use a variant of Theorem 4.1 that does not "union-bound over all chunks" since we are interested in the flow allocation of one chunk only. One easily verifies (using variable names as above):

Lemma 5.3: For a fixed chunk $\omega \in [n/\alpha]$, the ω-flow has capacity $W = \mu - L$ where $\mu = (1+\lambda_*)\alpha$ and:

$$L = O\left(\varphi\mu^{1/2}\Big(\log\frac{\varphi\mu}{\theta_*}\Big)^{1/4}\Big(\big(\mu/\varphi\big)^{1/4} + \varphi^{1/2}\Big)\right)$$

with probability no less than $1-\theta_*/3$.

Consequently, inequality (4) holds as long as $W \ge f$, which asymptotically resolves to $\alpha = O\left((\varphi^3/\lambda_*^4)\log(\varphi/\theta_*\lambda_*)\right)$. (The details of resolving $W \ge f$ are deferred to the Appendix.) ∎

Combining the Precode and the Code: As before, let $\lambda > 0$ be the desired overhead of the Practical Chunked Code (i.e., the combined Linear-time Chunked Code with a precode). It is straightforward to verify that when the precode parameters are as above, and the Chunked Code parameters are $\theta_* = (\lambda/4)/(1+\lambda)$ and $\lambda_* = \lambda/2$, the combined code design fulfills the desired requirements.

It is worth nothing that since the error probability of the Chunked Code vanishes exponentially, the combined code inherits the error probability of the precode. Therefore a stronger precode immediately results in a stronger combined code.

VI. Discussion

A. Adversarial Schedules

Previous work has studied network coding in the context of packet networks (see, e.g., [1], [2], [3], [4]). In much of this work, the packets lost on the network channels are modeled in the natural manner—as a stochastic process. In this paper, we have adopted a different approach, which we call adversarial schedules. Our model assumes that the successful packet transmissions are chosen by an adversary who knows our coding scheme but not its random choices. Additionally, packets may also be reordered and experience arbitrary delays.

Our adversarial model is strictly stronger than the aforementioned stochastic models, but yet the results that one can derive in this stronger model are essentially identical to those that can be derived in many weaker models (e.g., simple models based on Bernoulli or Poisson packet injections with i.i.d. losses). An explanation for this phenomenon is that the schedules typically generated by those stochastic models are in fact schedules that are intuitively "hard" for network coding schemes.

Thus our results based on adversarial schedules are not much stronger in a technical sense than those that one could derive with stochastic channels (other than, perhaps, the issue of reordered packets). The key benefit, we believe, of adversarial schedules is that one can eliminate the randomness of the channel from the analysis of the coding scheme (which is itself random). The resulting analysis is, we believe, clearer both conceptually and notationally.

B. Coding Delay

The notion of coding *delay* has been discussed in [17] as a primary characteristic of a coding scheme. (The traditional notion of rate is an other example of a primary characteristic that they consider.) We take a moment to explain how adversarial schedules relate to delay.

In brief, [17] describes a network environment as a (deterministic or probabilistic) set of transmission opportunities. For example, in the Discrete Bernoulli model a transmission opportunity is present at each clock tick between any pair of connected nodes. It is the job of the coding scheme to decide whether to utilize a transmission opportunity. Should a transmission opportunity be utilized, a packet is generated and sent. It may or may not be received depending on whether it is lost during transmission (according to the link's erasure behavior). If the packet is received, the transmission opportunity has been "utilized successfully." Notice that, traditionally, the job of a coding scheme is restricted to the packet generation (and the decoding at the sink), not the utilization choices.

The set of successful utilizations are, by definition, an adversarial schedule. Roughly, a coding scheme is called "capacity-achieving" if the flow capacity that it needs to deliver the message successfully approaches the message size asymptotically. It is important to notice how this definition of capacity is completely decoupled from the transmission opportunity utilization choices of the coding scheme. One can envision schemes that forgo plenty of opportunities, and in fact [17] explains that this makes it easier to achieve capacity. The notion of coding delay captures the extent to which a coding scheme forgoes transmission opportunities.

To summarize this discussion: the language of adversarial schedules is complementary to the notion of delay.

C. Is practical really practical?

We have already observed that one qualitative difference between Practical Chunked Codes and the state-of-the-art Raptor Codes is the polynomial dependence on the overhead in the former versus the poly-logarithmic in the latter. Is this difference significant?

A major difficulty with using Raptor Codes in some practical scenarios is that its encoding and decoding algorithms are rather "cache-unfriendly". This issue precludes Raptor Codes from being applied to very large messages (to the order of a few hundred gigabytes and more). Chunked coding, on the other hand, has some beneficial characteristics that may help in these scenarios. One useful characteristic is: during decoding, the data exhibits some locality properties. For example, the data needed to decode a single symbol also suffice to decode that symbol's entire chunk.

A detailed investigation of these performance issues is beyond the scope of the current paper. These issues are the focus of an ongoing research project [13] that focuses on optimizing Chunked coding for real-world performance. In [13], the Chunked coding scheme is generalized in that an input message of k symbols is associated with k chunks (instead of only k/α), such that the i^{th} chunk starts at the i^{th} input symbol and comprises a contiguous block of α consecutive input symbols. The encoding and decoding algorithms are modified appropriately. This coding scheme is dubbed *Smooth Perpetual Coding*. Roughly speaking, Smooth Perpetual codes try to ensure that the matrix of coding coefficients is banded, and thereby obtain efficient decoding algorithms.

Experimental results indicate that Smooth Perpetual Codes require apertures that are orders of magnitude smaller than their Chunked coding equivalents. In particular, a Smoothed Perpetual Code (in place of a Linear Chunked Code) with $\alpha = 32$ achieves approximately near-Raptor Code performance (using comparable implementations of both). Nevertheless, the usefulness of Smooth Perpetual Codes is still contingent upon the existence of a special kind of precode, described in [13] as an open problem.

ACKNOWLEDGEMENTS

We thank Yunnan Wu for several helpful discussions.

REFERENCES

[1] D. S. Lun, M. Médard, and M. Effros, "On coding for reliable communication over packet networks," in *Proc. 42nd Annual Allerton Conference on Communication, Control, and Computing*, Sept./Oct. 2004, invited paper.

[2] D. S. Lun, M. Médard, R. Koetter, and M. Effros, "Further results on coding for reliable communication over packet networks," in *Proc. 2005 IEEE International Symposium on Information Theory (ISIT 2005)*, Sept. 2005, pp. 1848–1852.

[3] Y. Wu, "A trellis connectivity analysis of random linear network coding with buffering," in *Proc. 2006 IEEE International Symposium on Information Theory (ISIT 2006)*, July 2006.

[4] D. S. Lun, M. Médard, R. Koetter, and M. Effros, "On coding for reliable communication over packet networks," submitted to *IEEE Trans. Inform. Theory*. [Online]. Available: http://arxiv.org/abs/cs.IT/0510070

[5] M. Wang and B. Li, "How practical is network coding?" in *Proc. 14th IEEE International Workshop on Quality of Service (IWQoS 2006)*, June 2006, pp. 274–278.

[6] P. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. 41st Annual Allerton Conference on Communication, Control, and Computing*, Oct. 2003, invited paper.

[7] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *Proc. 24th IEEE Conference on Computer Communications (INFOCOM)*, 2005.

[8] R. W. Yeung, "Avalanche: A network coding analysis," preprint. [Online]. Available: http://iest2.ie.cuhk.edu.hk/~whyeung/publications/Avalanche.pdf

[9] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inform. Theory*, vol. 52, no. 6, pp. 2551–2567, June 2006.

[10] P. Maymounkov, "Online codes," NYU, Technical Report TR2002-833, Nov. 2002.

[11] M. Luby, "LT codes," in *Proc. 43rd Annual IEEE Symposium on Foundations of Computer Science*, Nov. 2002, pp. 271–280.

[12] D. S. Lun, N. Ratnakar, M. Médard, R. Koetter, D. R. Karger, T. Ho, E. Ahmed, and F. Zhao, "Minimum-cost multicast over coded packet networks," *IEEE Trans. Inform. Theory*, vol. 52, no. 6, pp. 2608–2623, June 2006.

[13] P. Maymounkov, "Perpetual codes: Cache-oblivious coding," *In preparation*, 2006.

[14] N. Alon and J. H. Spencer, *The Probabilistic Method*, 2nd ed. Wiley Interscience, 2000.

[15] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical loss-resilient codes," in *STOC '97: Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997, pp. 150–159. [Online]. Available: citeseer.ist.psu.edu/luby98practical.html

[16] P. Oswald and A. Shokrollahi, "Capacity-achieving sequences for the erasure channel," 2002. [Online]. Available: citeseer.ist.psu.edu/oswald00capacityachieving.html

[17] P. Pakzad, C. Fragouli, and A. Shokrollahi, "Coding schemes for line networks," in *Proc. IEEE International Symposium on Information Theory*, 2005.

APPENDIX

DETAILS OF COROLLARY 4.3

We begin by fixing $q = k/\log^2 k$, $l_{\max} = O(1)$ and $p = O(1)$. By retracing the derivation of Theorem 4.2 one can see that with probability at least $1-\epsilon/2q$ the ω-flow for any fixed $\omega \in [q]$ is:

$$W = \frac{n}{q} - O\left(\left(\frac{n}{q}\right)^{3/4} c^{1/2}\right) \tag{6}$$

where $c = O(\sqrt{\ln n} + \sqrt{\ln 1/\epsilon} + \sqrt{\ln q})$. (Here c corresponds to $c' + c''$ from Theorem 4.2's proof.)

On the other hand, according to Theorem 3.9 each chunk ω must receive at least:

$$f = \frac{k}{q} + O(\ln k + \ln 1/\epsilon + \ln q) \tag{7}$$

flow in order to recover with probability no less than $1-\epsilon/2q$.

Therefore, whenever:

$$W \geq f \tag{8}$$

holds for all chunks, the entire input message is recovered with probability at least $1-\epsilon$ (using a union bound over all q chunks). When $\epsilon = 1/k^c$ for any constant c, inequality (8) resolves to $n = k + O(k/\ln^{1/4} k)$.

DETAILS OF THEOREM 5.2

The inequality $W \geq f$ resolves to:

$$(1+\lambda_*)\alpha - O\left(\varphi\mu^{1/2}\ln^{1/4}\frac{\varphi\mu}{\theta_*}\left(\left(\frac{\mu}{\varphi}\right)^{1/4} + \varphi^{1/2}\right)\right) \geq \alpha + O\left(\varphi\ln\frac{\alpha\varphi}{\theta_*}\right) \tag{9}$$

Observing that inside the $O(\cdot)$ notation μ is equivalent to α and unfolding, we get:

$$\lambda_*\alpha \geq O\Bigg(\underbrace{\varphi\ln\frac{\alpha\varphi}{\theta_*}}_{A} + \underbrace{(\varphi\alpha)^{3/4}\ln^{1/4}\frac{\varphi\alpha}{\theta_*}}_{B} + \underbrace{\varphi^{3/2}\alpha^{1/2}\ln^{1/4}\frac{\varphi\alpha}{\theta_*}}_{C}\Bigg) \tag{10}$$

We now examine each of the three terms on the right separately with respect to the left side. We can do this due to the $O(\cdot)$ notation. Additionally, we will employ the following lemma multiple times:

Lemma 1.1: For $x \geq 0$, the inequality $x \geq c\ln x$ is satisfied for all $x = \Omega(c\ln c + c\ln\ln c)$.

Term A resolves to:

$$\alpha \geq O\left(\frac{\varphi}{\lambda_*}\ln\alpha\right) \quad \text{and} \quad \alpha \geq O\left(\frac{\varphi}{\lambda_*}\ln\frac{\varphi}{\theta_*}\right) \tag{11}$$

Both conditions are met when:

$$\alpha \geq O\left(\frac{\varphi}{\lambda_*}\ln\frac{\varphi}{\lambda_*\theta_*}\right) \tag{12}$$

Term B resolves to:

$$\alpha \geq O\left(\frac{\varphi^3}{\lambda_*^4}\ln\alpha\right) \quad \text{and} \quad \alpha \geq O\left(\frac{\varphi^3}{\lambda_*^4}\ln\frac{\varphi}{\theta_*}\right) \tag{13}$$

One verifies that both conditions are met when:

$$\alpha \geq O\left(\frac{\varphi^3}{\lambda_*^4}\ln\frac{\varphi}{\lambda_*}\right) \tag{14}$$

Term C is superseded by term B, i.e. whenever α is bigger than term B (with respect to the $O(\cdot)$ notation) it is also bigger than term C.

Finally, inequalities (12) and (14) are combined in the single condition:

$$\alpha \geq O\left(\frac{\varphi^3}{\lambda_*^4}\ln\frac{\varphi}{\lambda_*\theta_*}\right) \tag{15}$$