Achieving Optimal Block Pipelining in Organized Network Coded Gossip

Majid Khabbazian, Di Niu
Department of Electrical and Computer Engineering
University of Alberta
mkhabbazian@ualberta.ca, *dniu@ualberta.ca*

Abstract—We use randomized network coding (RNC) with simple connection topology control to approach the theoretical limit on finish time of disseminating k blocks in a server cluster of n nodes. Unlike prior gossip literature which relies on completely random contact, we prove that with RNC, any receiver selection following a simple permutation rule can achieve a broadcast completion time of $k+n$ and that a time-varying random ring topology achieves a completion time of $k+o(k)+O(\log n)$, both with high probability. Since the theoretical limit on finish time is $k + \lceil \log_2 n \rceil$, our simple permutation algorithms achieve absolutely optimal (not only order-optimal) block pipelining for the k blocks. Our results hold for both one-to-all (broadcast) and all-to-all transfers. We demonstrate the usefulness of the proposed organized network coded gossip with an application to content distribution in cluster computing systems like MapReduce, and discuss practical block dividing strategies to hide the negative effect of computation overhead of network coding.

Index Terms—network coding; gossip; broadcast finish time; optimal block pipelining; computer cluster; content distribution; randomized algorithms

1 Introduction

Gossip protocols have extensive applications in content distribution, by disseminating data blocks through a randomized flooding process that guarantees information delivery to all the nodes with high probability. Since gossip does not rely on a stable network topology and incurs no structure maintenance overhead, it is potentially suitable for large-scale and robust content dissemination, where tree-based solutions are either costly to deploy or prone to failures. However, in contrast to structured multicast which guarantees the optimal use of network resources via spanning tree packing, traditional gossip can not guarantee the highest possible data throughput. In a classical time-slotted model, where a single node wants to disseminate k blocks to $n-1$ other nodes, and each node can upload at most 1 block to another node and download at most 1 block from another node per round, the optimal finish time is known to be $k+\lceil \log_2 n \rceil$ and can be achieved by a fully centralized sender-receiver pairing schedule and block selection strategy [1]. Nonetheless, no decentralized random gossip is known to exactly approach this limit, although *order-optimal* results are derived (e.g., [2]).

Network coding promises to improve dissemination efficiency in multi-block gossips, as the challenge of block selection is replaced by a random block mixing scheme, namely, random linear network coding (RLNC) in a finite field. However, optimal pipelining in k is still not achieved. In a random phone call model [3], if each node holds one of the k blocks initially, and in each round transmits a coded block to a random receiver, the finish time is $ck + O(\sqrt{k}\log(k)\log(n))$, where c is a constant between $3-6$. This bound is further tightened in [4], which proves a finish time of $k+o(k)$. Such optimal pipelining in k, however, is only achieved when each node holds a subset of k blocks initially. For the case of a single source node, achieving optimal pipelining in k with gossip algorithms is to be explored yet.

In this paper, we show that by using a simple sender-receiver pairing rule [5] instead of random phone call, RLNC can almost approach the limit of $k + \lceil \log_2 n \rceil$, achieving absolutely optimal (*not only order-optimal*) block pipelining in k, no matter whether all nodes, a subset of nodes or only a single source node holds some blocks initially. The rule only requires that all the nodes form an arbitrary or random ring topology (permutation) in each time slot, and each node transmits a coded block to its immediate successor on the ring, complying with both the upload and download constraints of 1 block per round. We show that the worst-case finish time is $k+n$ with arbitrary permutations, and is $k + o(k) + O(\log n)$ with random permutations, both with high probability.

We demonstrate the usefulness of the proposed network coded gossip with application to frequent data transfers in computing clusters. Data transfers could constitute performance bottlenecks in computing clusters running MapReduce [6], Spark [7] and Dryad [8], where large amounts of data must be transferred among servers (e.g., from mappers to reducers) between computation stages [9].

While standing between fully random gossip and structured multicast, the permutation gossip becomes a natural efficient solution to content distribution tasks in

controllable server clusters. On one hand, unlike a peer-to-peer (P2P) system formed by unreliable end-hosts, in a server cluster, one has full control of server connection topology and can arrange it easily at any time. On the other hand, due to the data size and frequently changing transfer objectives, it is too costly and failure-prone to maintain tree-like structures or perform centralized block-level scheduling. Moreover, cluster servers are usually geographically collocated in enterprises, institutions or datacenters with similar bandwidth and processing capacity, justifying the effectiveness of simple bandwidth allocation schemes.

Finally, the fact that the number of nodes n is no more than a few dozens in a computing cluster justifies our focus on achieving block-level pipelining in k which can grow quite large, although we also rigorously analyze the dependence of finish time on n and the field size q. In particular, we show that even in the case of XOR operations ($q = 2$), the finish time is no more than $2k + f(n)$, as k scales.

Our simulation results show that the proposed semi-organized network coded gossip can finish broadcasting k blocks in no more than $k + \lceil \log_2 n \rceil + 4$ rounds, which is at most 4 more rounds than the theoretical limit $k + \lceil \log_2 n \rceil$ and strongly supports our theoretical derivation of $k + o(k) + O(\log n)$ rounds. More importantly, we carefully simulate the computation overhead of network coding, and propose a simple 3-step (receiving/coding/sending) pipelining approach at each node to hide the negative effect of coding latency, as long as it is confined to less than the block transfer latency. Finally, based on measurements collected from the simulation, we provide an effective method to estimate the total data transfer time of our algorithm in reality, and discuss optimized block dividing schemes given the data to be transferred and system configurations.

The remainder of the paper is organized as follows. In Sec. 2, we review the related literature. In Sec. 3, we present some motivating examples of data transfers in cluster computing systems before describing our system model. In Sec. 4, we introduce the random network coding scheme and the permutation-based topology control algorithms used in the paper. We provide the performance bounds on the finish time (in terms of rounds) of arbitrary permutation with RLNC and random permutation with RLNC, in Sec. 5 and Sec. 6, respectively. In Sec. 7, we present simulation results based on the encoding of real packets to verify our theoretical findings. In the same section, we also propose a 3-step pipelining technique to reduce the negative effect of coding latency and discuss the optimal block dividing strategies given the size of the file. The paper is concluded in Sec. 8.

2 Relationship to Prior Work

A common objective of content dissemination protocols is to minimize the finish time, subject to node capacity constraints. Traditional solutions are mostly based on multiple trees. Chiu *et al.* [10] and Li *et al.* [11] show that packing only depth-1 and depth-2 spanning or Steiner trees can achieve the maximum feasible broadcast rate limited by the node download capacity constraints. The rates are then assigned to the constructed multi-trees to maximize the data streaming rate. Chen *et al.* [12] further describe a primal-dual algorithm for the distributed implementation of tree construction and rate allocation. However, tree-based algorithms remains largely centralized with non-trivial structure maintenance overhead. Even in distributed implementation, it is not certain if the algorithm can converge with the presence of node dynamics or random failures. The frequently changing dissemination tasks in a server cluster also makes a tree-solution cumbersome to manage.

Gossip algorithms, on the other hand, enjoys the superior adaptability to arbitrary network and topology and the simplicity of implementation. Sanghavi *et al.* [2] show the order-optimality of non-coding gossiping, with the aid of node buffer state exchanges. With node state reconciliation, Massoulie *et al.* [13] prove the strict optimality of non-coding block selection protocols. However, it is theoretically shown in [14] that, a common problem with gossip protocols that require state exchanges is that their success heavily depends on the accuracy and frequency of state updates. As compared to these works, the use of network coding completely eliminates the need of node buffer exchanges, further reducing the block scheduling overhead that escalates as k grows.

Deb *et al.* [3] show the order-optimality of network coding gossip in a random phone call model, assuming k blocks are spread across the network to start with. Haeupler [4] further proves that a finish time of $k + o(k)$ is achieved in the same model for a random pull protocol when each node holds a subset of k blocks initially. For the case that only one source node wants to broadcast its blocks to all other nodes, optimal pipelining in k is not known yet. Shifting away from the random phone call model, we provide a stronger result that a class of sender-receiver pairing rules satisfying a simple permutation condition can achieve absolutely optimal pipelining in k regardless of the initial states of nodes. Such a permutation is easily implemented in any controllable server clusters. To keep the coding complexity and coefficient overhead low, we show that even in a binary field, the finish time at most doubles the optimal finish time when k is large.

Finally, different from previous works [2], [3], [10]–[13], [15], [16] that assume unbounded node download capacities, in this paper, we consider both node upload and download bandwidth constraints.

3 Motivating Examples and the System Model

In big data computing clusters, both one-to-many and many-to-many data transfers among servers take place

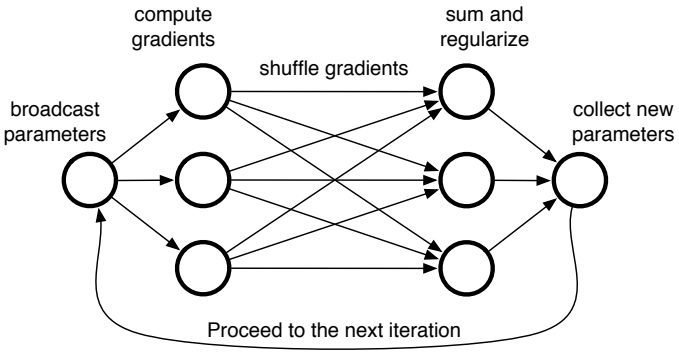


Fig. 1. The workflow of a logistic regression algorithm in a computing cluster. Each circle represents a server. Courtesy to [9].

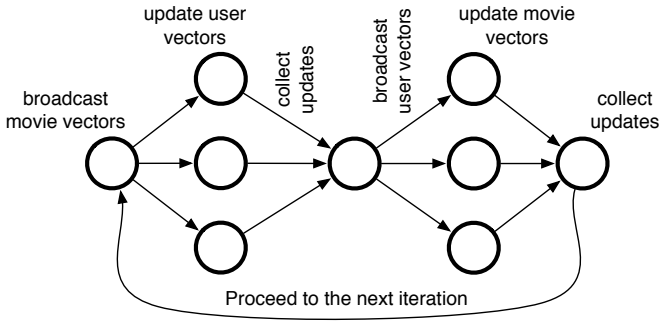


Fig. 2. The workflow of a collaborative filtering algorithm in a computing cluster. Each circle represents a server. Courtesy to [9].

frequently. We describe a few of such scenarios before presenting our mathematical model.

MapReduce Systems. MapReduce [6] is a parallel programming framework to process large data sets on a computing cluster. A MapReduce program comprises map tasks, each running on a server to process a part of the data in parallel, and reduce tasks, each running on a server to perform a summary operation based on the availability of map task outputs. In such systems, each reducer needs to acquire the outputs from one, multiple, or even all the map tasks. Data transfers between mappers and reducers in a MapReduce framework (usually referred to as shuffle) can be modelled as a many-to-many transfer. Measurements [9] have verified that data transfers can account for more than 50% of the total running time in jobs that have reduce tasks, based on an analysis of a week-long trace dataset from Facebook's Hadoop cluster. Therefore, it is highly desirable to devise effective algorithms to alleviate the data transfer bottleneck [17] widely reported in MapReduce.

Iterative Algorithms on Spark. As an example of iterative learning algorithms performed on a MapReduce framework, Monarch [18] is an application to identify spam links on Twitter. For each tweet, features related to the page linked to (e.g., IP address, domain name, and frequencies of words on the page) are collected as the inputs to Monarch. Given the features of a large amount of tweets, Monarch identifies which features are highly correlated with spam links using logistic regression [19] and is implemented on *Spark* [7], an iterative MapReduce computation framework that provides fault-tolerant distributed collections.

The work flow of the logistic regression is depicted in Fig. 1. Each iteration includes a large broadcast of $200 - 400$ MB and a shuffle of 10000 MB per reducer. Each data transfer forms a bottleneck of the entire job, which cannot proceed before the last node finishes in the corresponding data transfer. Measurements [9] show that data transfer can account for 42% of the total algorithm running time, with 30% spent in broadcast and 12% spent in shuffle on a 30-node cluster. Apparently, optimizing the finish times of these transfers is critical to the efficiency of iterative learning algorithms.

Collaborative Filtering. Collaborative filtering has been used to solve the Netflix Challenge of movie recommendation on a computing cluster [20], with the goal of predicting user ratings for movies that they have not seen based on their ratings for other movies. Each user and each movie are modelled as feature vectors. A user's rating for a movie is predicted as the dot product of the feature vector of the user and that of the movie. These vectors can be found through an iterative process, illustrated in Fig. 2. The algorithm broadcasts the user or movie feature vectors alternately, allowing the (server) nodes to optimize the other set of vectors in parallel. Each broadcast phase can transfer several hundred megabytes of data, seriously delaying the algorithm execution. With 60 nodes, the data transfers use 45% of the algorithm running time [9].

For details of above motivating examples, interested readers are invited to read [7], [9], [18]–[20].

3.1 The System Model

We present a theoretical model that allows us to study content distribution in clusters. We model the server cluster over which data dissemination is performed as a network of n nodes, where each node can transmit packets to any other node through a TCP connection. The data to be disseminated is divided into k uniform-sized *blocks*. (The actual dividing strategies will be studied in Sec. 7.) We consider a time-slotted model, where data transfer happens in rounds. We assume each node can send (upload) at most 1 data block to another node and receive (download) at most 1 data block from another node in each round. The objective is to disseminate all k blocks to all n nodes no matter what the initial state is. Depending on the initial states, we consider two scenarios: *one-to-all* transfer or *broadcast*, where only a single source node holds all the blocks initially, and *many-to-all* transfers, where each node may hold some blocks initially.

The above model is more suitable to a computing cluster than a P2P network or the Internet. Nodes in

a computing cluster usually have similar configurations in terms of CPU, memory and network I/O. The servers installed in a batch can be viewed as possessing homogeneous capacity and bandwidth provisioning. Furthermore, cluster nodes are often collocated geographically in enterprises, campus networks or datacenters, making the estimation of network environments and the reservation of bandwidth easier. These facts eliminate the need for more sophisticated bandwidth allocation schemes (such as offered by tree-packing) as required in a much larger P2P network and justifies the feasibility of a simple gossip-like protocol as above.

The above model can capture the characteristics of both broadcast and shuffle transfer tasks in cluster computing. The one-to-all transfer exactly models the broadcast in the logistic regression and collaborative filtering examples. The many-to-all transfer is an extreme case of the shuffle examples mentioned above. In shuffle tasks, there are multiple source nodes each holding some blocks. If each source node only needs to distribute its data to a subset of all nodes, the finish time will be upper-bounded by the finish time when it disseminates its data to all nodes. The latter scenario is exactly the many-to-all transfer in our model.

It is worth noting that although our target application is data transfers in computing clusters, there is still a large gap between our round-based theoretical model and the real scenario. However, since the focus of this paper is on theory, it represents a first step towards investigating the feasibility of using network coded gossip to speedup data transfers in clusters.

4 ALGORITHMS

In this section, we describe the random network coding scheme and the connection topology control algorithms used in the paper.

4.1 Random Linear Network Coding

When nodes adopt the RLNC algorithm [21], the source node(s) send out linear combinations of the original blocks that can initially be possessed by one or multiple nodes. Network coding is allowed in the sense that when each node transmits a block, it should use all the blocks, whether they be original or received coded blocks, that the node possesses so far to generate a coded block. Therefore, the coded blocks sent in the network are linear combinations of the k original data blocks. All the (coded and original) blocks can be represented as vectors over a finite field $\mathbb{F}_q$, where q is a prime or power of a prime. We assume each of the k original blocks $\vec{b}_1, ..., \vec{b}_k$ is a vector from $\mathbb{F}_q^l$ of length l. Hence, each coded block sent around in the network can be represented as a tuple $(\vec{c}, \vec{b})$, where $\vec{c} = (c_1, \ldots, c_k) \in \mathbb{F}_q^k$ are the coefficients and $\vec{b} = \sum_{i=1}^{k} c_i \vec{b}_i \in \mathbb{F}_q^l$ is the payload as a linear combination of original blocks. In other words, each coded block only records a payload $\vec{b}$ together with the coefficients $\vec{c}$ needed to generate $\vec{b}$ from the original blocks. To decode the original blocks using Gaussian elimination, a node only needs to have k blocks with linearly independent coefficient vectors.

Apparently, a node can produce any block whose coefficient vector is spanned by the coefficient vectors of the blocks it already has. If the coefficient vectors of the blocks a node has can span the full space $\mathbb{F}_q^k$, the node can literally decode. It is worth noting that whether a node can decode or not is only determined by the coefficient vectors it has, but not the payload. Therefore, we can solely focus on the spreading of coefficients. Following Haeupler's [4] analysis technique, we describe the RLNC algorithm in the following manner:

RLNC: Each node v maintains a subspace X_v that is the span of all the coefficient vectors the node has. If v does not have any block initially, X_v is initialized to contain only the zero vector. If v has some block(s) $\vec{b}_i$ initially, X_v is initialized to contain the ith standard basis vector(s). X_v also contains all linear combinations that complete the span of these vectors. When node v sends a block, it chooses a uniformly random coefficient from X_v to generate the coded block. At the end of each round, X_v is updated with the new coefficients received by node v. If the subspace X_v spanned by the coefficient vectors is the full space $\mathbb{F}_q^k$, then node v can decode all k original blocks.

Note that both the coefficient overhead (the additional block header to store coefficients) and coding complexity increases with q. Fortunately, modern servers can efficiently encode and decode blocks on-the-go in $GF(2^8)$ [22], with $8k$ bits of coefficient overhead in each (coded) block. If $k = 200$ blocks, each of 1 MB, are to be transferred, the coefficient in each takes 200 bytes which represents $200\text{B}/1\text{MB} = 0.02\%$ overhead.

4.2 Permutation-based Receiver Selection

A heavily studied model in gossip literature is the random phone call model [3], where each node selects a random node as the receiver (or the sender) to push (or pull) a block. However, little result is known for random phone call models to achieve absolutely optimal pipelining in terms of k. In this paper, we study in depth a new class of algorithms based on permutation that first appeared in [5]:

Random and Arbitrary Permutation: In each round, a permutation, $u_1, u_2, \ldots, u_n$ of all n nodes is formed, such that node u_i, $1 \leq i < n$, sends a block to node u_j where

$$j = i + 1 \mod n.$$

If a random permutation is chosen uniformly from all permutations, we call the algorithm *random permutation*. If an arbitrary permutation (any possible permutation, including one contrived by an adversary) is chosen, we call the algorithm *arbitrary permutation*.

Although the permutation algorithms mentioned above are harder to realize in a fully distributed fashion than random phone call, it can be easily suited to the server cluster scenario, where we have full centralized control of the connection topology. Moreover, the homogeneous nature of the server capacities in a computing cluster further justifies the feasibility of forming a permutation among all nodes. On the other hand, with permutation-based receiver selection, the data transmission is still fully distributed using RLNC. Therefore, our algorithms act as a compromise between fully decentralized gossip and fully centralized spanning tree packing, taking advantage of the fact that we have full control of topologies in a computing cluster, while avoiding the hassle of structure maintenance in massive content distribution. In this respect, our algorithms can be deemed as "semi-distributed."

5 Finish Time of Arbitrary Permutation with RLNC

In this section, we provide performance bounds on the finish time (in terms of rounds) of arbitrary permutation with RLNC, as described above. We not only prove absolutely optimal pipelining (instead of order-optimal) results of the finish time as a function of the number of blocks k, but also characterize the dependency of the finish time on the number of nodes n and the field size q.

Our analysis technique depends on an important notion of *knowing* first proposed by Haeupler [4]: a node A is defined as knowing about $\vec{\mu} \in \mathbb{F}_q$ if its coefficient subspace X_A is not orthogonal to $\vec{\mu}$, i.e., if there is a vector $\vec{c} \in X_A$ with $< \vec{c}, \vec{\mu} > \neq 0$. A node A knowing a vector $\vec{\mu}$ does not imply $\vec{\mu} \in X_A$ or that A is able to decode a block associated with the coefficients $\vec{\mu}$. Knowing $\vec{\mu}$ only indicates that node A is not completely ignorant about $\vec{\mu}$. However, two useful facts about knowing in the following lemma will help us prove tight bounds for our algorithms with the aid of tail inequalities.

Lemma 1 (Haeupler [4] (2011))**.** *If a node A knows about a vector $\vec{\mu}$ and transmits a block to node B, then B knows about μ afterwards with probability at least $1 - 1/q$. Furthermore, if a node knows about all the vectors in $\mathbb{F}_q^k$, then it is able to decode all k original blocks.*

Let us now analyze the finish time of *arbitrary* permutation with RLNC, as a function of n, k and q. In particular, when the field size q is large enough, we provide a very tight upper bound of $k+n$ on the worst-case finish time. Furthermore, even if q is as small as 2, the worst-case finish time is in the form of $2k + f(n)$.

We characterize the dependency of worst-case finish time on n, k and q in Theorem 2 using Lemma 1. Based on Theorem 2, we provide corollaries to show the finish time behavior under different parameter choices.

Theorem 2. *Let r be a positive integer. With probability at least $1 - \epsilon$, every node will receive all k blocks in*

$$k + (n-1) + r$$

rounds, if $q \geq q_{min}$, where

$$q_{min} = \left(\frac{1}{\epsilon} \cdot \binom{k + (n-1) + r}{k + r} \right)^{\frac{1}{r}}.$$

The above result holds regardless of initial states of the nodes or the choice of permutation in each round.

Proof: The basic idea of our proof is to give the probability that every node knows a specific vector $\lambda \in \mathbb{F}_q^k$, and then use a union bound to bound the probability that every node knows all q^k vectors in $k + (n-1) + r$ rounds. We first provide the following trivial lemmas without proof which will be useful in the detailed proof of Theorem 2:

Lemma 3. *Let m and t be positive integers, and $Y_1, Y_2, \ldots, Y_{m+t}$ be a collection of $m + t$ i.i.d Bernoulli variables with $\Pr(Y_i = 1) = p$ for every $1 \leq i \leq m + t$. Let $Y = \sum_{i=1}^{m+t} Y_i$. Then*

$$\Pr(Y < m) \leq \binom{m+t}{t+1} (1-p)^{t+1}.$$

Lemma 4 (Khabbazian *et al.* [23] (2011))**.** *For every positive integer i, let X_i and Y_i be $\{0,1\}$-valued random variables. Suppose that $\{Y_i | i = 1, 2, \ldots\}$ is a collection of independent random variables. Suppose further that:*

1) $\Pr(X_1 = 1) \geq \Pr(Y_1 = 1)$.
2) *For every $j \geq 2$ and $x_1, x_2, \ldots, x_{j-1} \in \{0,1\}$, $\Pr(X_j = 1 | X_1 = x_1, , X_{j-1} = x_{j-1}) \geq \Pr(Y_j = 1)$.*

Then for every integer $t \geq 1$ and every nonnegative integer d,

$$\Pr\left(\sum_{i=1}^{t} X_i \geq d \right) \geq \Pr\left(\sum_{i=1}^{t} Y_i \geq d \right).$$

Now we can prove Theorem 2. We prove that with probability at least $1 - \epsilon$, every node knows all the vectors. Then by Lemma 1, every node will be able to decode k blocks. Let us fix a vector λ. Let X_i be a random {0,1}-random variables such that $X_i = 1$ if and only if i) the number of nodes knowing λ at the end of round i increases, or ii) every node knows λ by the end of round i. In every round, if there is at least a node that does not know λ, we can find two nodes u and v such that u sends a packet to v in that round, and u knows λ but v does not. The probability that, by the end of that round, v knows λ is $1 - \frac{1}{q}$. Therefore, the number of nodes knowing λ increases by probability at least $1 - \frac{1}{q}$, if there is at least one node that does not know λ. Consequently, for every $x_1, x_2, \ldots, x_{j-1} \in \{0,1\}$, we have $\Pr(X_i = 1 | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) \geq 1 - \frac{1}{q}$. Let Y_i be a collection of independent Bernoulli variables such that $\Pr(Y_i = 1) = 1 - \frac{1}{q}$. By lemma 4, we have

$$\Pr\left(\sum_{i=1}^{k+(n-1)+r} X_i \geq n \right) \geq \Pr\left(\sum_{i=1}^{k+(n-1)+r} Y_i \geq n \right). \quad (1)$$

Replacing m with n, t with $k+r-1$, and p with $1-\frac{1}{q}$ in Lemma 3, we get

$$\Pr\left(\sum_{i=1}^{k+(n-1)+r} Y_i < n\right) \leq \binom{k+(n-1)+r}{k+r}\frac{1}{q^{k+r}}.$$

Thus, by (1), we get

$$\Pr\left(\sum_{i=1}^{k+(n-1)+r} Y_i \geq n\right) \geq 1-\binom{k+(n-1)+r}{k+r}\frac{1}{q^{k+r}}.$$

Consequently, by a union bound, the probability that in $k+(n-1)+r$ rounds, all nodes know all q^k vectors λ is at least

$$\begin{aligned}&1-\binom{k+(n-1)+r}{k+r}\frac{1}{q^{k+r}}\cdot q^k\\ =&1-\binom{k+(n-1)+r}{k+r}\frac{1}{q^r}\\ \geq&1-\epsilon\end{aligned}$$

where the last inequality is because $q \geq q_{min}$. □

Remarks. Theorem 2 provides a guideline on choosing the field size q given a target finish time and success probability. Increasing the field size q has two major effects: i) By Lemma 1, if a node A knows about a vector $\vec{\mu}$ and transmits a block to node B, then B knows about $\vec{\mu}$ with higher probability when q increases. In other words, a vector propagates faster if q is larger. ii) As q increases, a node must know more vectors in order to decode. Overall, the effect of the former is higher than the latter, hence nodes can decode earlier when q increases. If we allow q to be large enough, we can give a tight worst-case bound on the finish time of arbitrary permutation.

Corollary 5. *For a sufficiently large field size q, every node will receive all k blocks with high probability within $k+n$ rounds, regardless of the initial states of the nodes or the choice of permutation in each round.*

Proof: Setting $r=1$ in Theorem 2, we get

$$q_{min} = \frac{1}{\epsilon}\cdot\binom{k+n}{k+1}$$

Note that we have the followings inequalities:

$$\begin{aligned}&\binom{k+n}{k+1} \leq 2^{k+n},\\ &\binom{k+n}{k+1} \leq \left(\frac{e\cdot(k+n)}{k+1}\right)^{k+1},\\ &\binom{k+n}{k+1} = \binom{k+n}{n-1} \leq \left(\frac{e\cdot(k+n)}{n-1}\right)^{n-1}.\end{aligned}$$

Therefore, $\log_2 q_{min}$ is at most

$$\log_2(\frac{1}{\epsilon}) + \min\left\{n+k, (k+1)\log_2\left(\frac{e\cdot(k+n)}{k+1}\right), (n-1)\log_2\left(\frac{e\cdot(k+n)}{n-1}\right)\right\}$$

which gives a lower bound on the size of finite field q. □

Remarks. To see why the upper bound of $k+n$ on the finish time is very tight, we imagine an adversary that always forms a fixed line network

$$s \to 1 \to \ldots \to n-1$$

in all the rounds. Suppose that s is the single source, while other nodes hold no block initially. Under such a fixed permutation, we let blocks be streamed from s to $n-1$ along the line one by one. It takes k rounds for the kth (last) block to reach node 1, and another $n-2$ rounds to reach node $n-1$. This simple scheme achieves a finish time of $k+n-2$. Thus, the worst-case finish time of arbitrary permutation is between $k+n-2$ and $k+n$ rounds. We can almost conclude that the worst scenario one can encounter is that the permutation is fixed in all rounds and there is a single source. Since we have full control of the connection topology in a server cluster, if the permutation is changed in each round, the finish time could be much reduced.

However, the q_{min} required by Corollary 5 could get quite large. For example, if $n=10$ and $k=1000$, we get $\log_2 q_{min} \leq 75$ bits. A natural question to ask is — what if the field size q is set to a small value in order to reduce the coding complexity and the coefficient overhead in block headers? We have obtained the following result for $q=2$:

Corollary 6. *If $q=2$ then in*

$$k+(n-1)+\max\{k+n-1, 2n\log_2 2n\}$$

rounds, every node will receive all k blocks, with high probability[1]*, regardless of the initial states of the nodes or the choice of permutation in each round.*

Proof: By Theorem 2 it suffices to show that $q_{min} \leq 2$ if we set $r = \max\{k+n-1, 2n\log_2 2n\}$ and

$$\epsilon = \frac{1}{\left(\frac{n-1}{6}\right)^{n-1}\cdot(k+n-1)} \leq \frac{1}{\text{poly}\left(\max\{n,k\}\right)}.$$

1. By high probability, we mean with probability at least $1-\frac{1}{\text{poly}(\max\{n,k\})}$.

Using Theorem 2, we have

$$\begin{aligned} q_{min} &= \left(\frac{1}{\epsilon}\cdot\binom{k+(n-1)+r}{k+r}\right)^{\frac{1}{r}} \\ &= \left(\frac{1}{\epsilon}\cdot\binom{k+(n-1)+r}{n-1}\right)^{\frac{1}{r}} \\ &\le \left(\frac{1}{\epsilon}\cdot\frac{(k+(n-1)+r)^{n-1}}{(n-1)!}\right)^{\frac{1}{r}} \\ &\le \left(\frac{1}{\epsilon}\cdot\frac{(2r)^{n-1}}{(n-1)!}\right)^{\frac{1}{r}} \\ &\le \left(\frac{1}{\epsilon}\cdot\frac{2^{n-1}r^{n-1}}{\left(\frac{n-1}{3}\right)^{n-1}}\right)^{\frac{1}{r}} \\ &= \left(\frac{1}{\epsilon}\cdot\frac{r^n}{r\left(\frac{n-1}{6}\right)^{n-1}}\right)^{\frac{1}{r}} \\ &= (r^n)^{\frac{1}{r}} \\ &= r^{\frac{n}{r}} \\ &= 2^{\frac{n\log_2 r}{r}} \end{aligned} \tag{2}$$

Note that the function $f(x)=\frac{\log_2 x}{x}$ is non-increasing for $x>e$. Also, $r\ge 2n\log_2 2n > e$ since $n\ge 2$. Therefore,

$$\begin{aligned} \frac{n\log_2 r}{r} &\le \frac{n\log_2(2n\log_2 2n)}{2n\log_2 2n} \\ &= \frac{\log_2(2n\log_2 2n)}{2\log_2 2n} \\ &\le \frac{\log_2((2n)^2)}{2\log_2 2n} = 1, \end{aligned}$$

where the last inequality is by the fact that $\log_2 2n \le 2n$ for $n\ge 2$. Consequently, by (2), we get $q_{min}\le 2$. □

Remarks. Corollary 6 implies that even if coding operations are as simple as XORs in a binary field, as k grows, arbitrary permutation with RLNC can finish the transfer in at most $2k+f(n)$ rounds, where $f(n)$ is a function of n. Since in a server cluster, we usually have $k\gg n$, this means that even though XOR cannot achieve absolutely optimal pipelining in k, the loss in broadcast rate has a factor of at most 2.

In addition, all the results mentioned above do not rely on the initial states of the nodes, and thus hold for both broadcast and shuffle transfers. In the shuffle case, where each node holds at least some blocks initially, the finish time could be much lower than that of broadcast.

6 Finish Time of Random Permutation with RLNC

Although absolutely optimal pipelining in terms of k is achieved even with arbitrary permutation, the worst-case bound $k+n$ on the finish time is still far from the theoretical limit $k+\lceil\log_2 n\rceil$ which is achieved by fully centralized control and scheduling. In addition, the field size q required to achieve optimal pipelining is also large according to Corollary 5. Much tighter bounds in terms of n can be derived if the receiver selection is determined by random node permutation rather than arbitrary permutation (including that controlled by a strong adversary who is aware of the algorithm and states of nodes).

In this section, we show that the finish time of random permutation with RLNC is $k+O(\log n+\log k)$ with high probability, which is very close to the theoretical optimal value $k+\lceil\log_2 n\rceil$, not only in terms of the number of blocks k, but also in terms of the number of nodes n.

6.1 Main Result

Our main result regarding the finish time of random permutation with RLNC is presented in the following theorem:

Theorem 7. *Suppose, in every round, a permutation is selected uniformly at random. Let $q\ge n$. Then, every node will receive all k blocks in*

$$k+O(\log n+\log k)$$

rounds, with a high probability which is at least $1-\frac{1}{nk}$, regardless of the initial states of the nodes.

Remarks. Theorem 7 shows that if we replace arbitrary permutation with random permutation, the finish time is reduced significantly from $k+n$ down to $k+O(\log n+\log k)$, which is very close to $k+\lceil\log_2 n\rceil$. Even if the number of servers is as large as several hundreds, the $\log n$ term is almost negligible in the finish time. Furthermore, the assumption $q\ge n$ in Theorem 7 is easily warranted, since in a computing cluster like Hadoop and Spark, the number of servers is usually less than 50 and seldom exceeds 100 or 200 even for very large datasets. Even if we set $q=256=2^8$, which is a common practice in P2P networks [5], [22], [24], only 8 bits are needed to store each finite field element, incurring neglegible coefficient overhead.

6.2 Proving Theorem 7

In this section, we derive the rigorous proof for Theorem 7. Unlike in worst-case arbitrary permutation, tail inequalities must be carefully used to bound the random behavior of nodes. Before presenting the detailed proofs, let us first describe the high-level intuitions of our proof technique.

We divide the entire transfer process into 4 stages. For the first $\Theta(\log n)$ rounds, or the first stage, in every round the number of nodes knowing a vector $\vec{\mu}$ increases by at least one. We show that, in the second stage, where there are initially at least $\Theta(\log n)$ nodes knowing $\vec{\mu}$, the number of nodes knowing $\vec{\mu}$ increases exponentially in every round, w.h.p, until the majority of nodes know about $\vec{\mu}$. At the beginning of the third stage, the majority of nodes already know $\vec{\mu}$. In this stage, we show that the number of nodes not knowing $\vec{\mu}$ decays exponentially, in every round, w.h.p. until there are at most $\Theta(\log n)$ nodes

not knowing $\vec{\mu}$. This brings us to the last stage, at the beginning of which there are at most $\Theta(\log n)$ nodes not knowing $\vec{\mu}$. Since in every round, at least one node that does not know about $\vec{\mu}$ will know it, the fourth stage will last at most $\Theta(\log n)$ rounds. The $O(\log n)$ additive term in the theorem statement comes from the fact that each stage lasts $O(\log n)$ rounds, w.h.p. The $O(\log k)$ term is to increase the probability that a node knows about $\vec{\mu}$ to a level sufficient to show, using Chernoff and union bounds, that everyone knows about every vector w.h.p.

The following lemmas will be useful in the proof of our main result for random permutation:

Lemma 8. *Let $A \geq 1$ and $B \geq 1$ be real numbers. Let r be any real number such that*

$$r \geq A \log \left(2(A+B)\log(A+B)\right).$$

Then, we have

$$r \geq A \log(B+r).$$

Proof: Note that $r \geq A\log(4\log 2) \geq A$, because $A \geq 1$ and $B \geq 1$. Let $f(x) = x - A\log(B+x)$. We have

$$f'(x) = 1 - \frac{A}{B+x},$$

thus the function is non-decreasing for $x \geq A$. Therefore, to prove the lemma, it is sufficient to show that $r_0 \geq A\log(B+r_0)$ (that is $f(r_0) \geq 0$), where

$$r_0 = A \log \left(2(A+B)\log(A+B)\right).$$

Note that $x \geq 2\log(x)$, for every positive real number x. Hence,

$$\begin{aligned}
&(A+B) \geq 2\log(A+B)\\
&\implies \log(A+B) \geq \log(2\log(A+B))\\
&\implies 2\log(A+B) \geq \log(A+B) + \log(2\log(A+B))\\
&\implies 2\log(A+B) \geq \log\left(2(A+B)\log(A+B)\right)\\
&\implies 2A\log(A+B) \geq A\log\left(2(A+B)\log(A+B)\right) = r_0\\
&\implies B + 2A\log(A+B) \geq B + r_0\\
&\implies 2(A+B)\log(A+B) \geq B + r_0\\
&\implies \log(2(A+B)\log(A+B)) \geq \log(B+r_0)\\
&\implies A\log(2(A+B)\log(A+B)) \geq A\log(B+r_0).
\end{aligned}$$

Thus, we have $r_0 \geq A\log(B+r_0)$, proving the lemma. □

Lemma 9. *Let n, d_1, d_2 and d_3 be positive integers such that $d_1 \geq 2d_2$, and $d_2 \geq d_3 \geq 16\log n$. Let*

$$I = \left\{ [0, \frac{1}{d_2}), [\frac{1}{d_2}, \frac{2}{d_2}), \ldots, [\frac{d_2-1}{d_2}, 1] \right\}.$$

and I', $|I'| \geq d_3$, be a subset of I. Let P be a set of d_1 points selected uniformly and independently at random from $[0,1]$. Then, with probability at least $1 - \frac{1}{n^2}$, there are at least $\frac{d_3}{2}$ intervals in I' that contain at least a point from P.

Proof: Suppose points in P are selected from $[0,1]$ uniformly and independently at random in d_1 steps, one point per step. Let $I_i'' \subseteq I'$, $1 \leq i \leq d_1$, be the set of intervals in I' that contain at least a point by the end of ith step. We define $I_0'' = \emptyset$. Let X_i, $1 \leq i \leq d_1$, be $\{0,1\}$-valued random variables such that

$$X_i = 1, \text{if} |I_i''| > |I_{i-1}''| \text{or} |I_{i-1}''| \geq \frac{d_3}{2}.$$

Note that

$$\Pr(X_i = 1 | X_{i-1} = x_i, \ldots, X_1 = x_1) = 1$$

if $\sum_{j=1}^{i-1} x_j \geq d_3/2$, and

$$\Pr(X_i = 1 | X_{i-1} = x_i, \ldots, X_1 = x_1) \geq \frac{\lceil \frac{d_3}{2} \rceil}{d_2} \geq \frac{d_3}{2d_2},$$

if $\sum_{j=1}^{i-1} x_j < d_3/2$. Therefore, we have

$$\Pr(X_i = 1 | X_{i-1} = x_i, \ldots, X_1 = x_1) \geq \frac{d_3}{2d_2}.$$

Let Y_i, $1 \leq i \leq d_1$, be a set of d_1 i.i.d Bernoulli random variables with

$$\Pr(Y_i) = d_3/2d_2.$$

Let $X = \sum_{i=1}^{d_1} X_i$ and $Y = \sum_{i=1}^{d_1} Y_i$. By a Chernoff bound we have

$$\Pr(Y < \frac{\mu_y}{2}) \leq e^{-\frac{\mu_y}{8}}, \tag{3}$$

where $\mu_y = d_1 \cdot d_3/2d_2$ is the mean of Y. Since $d_1 \geq 2d_2$, we get $\mu_y \geq d_3$, hence, by (3), we have

$$\Pr(Y < \frac{d_3}{2}) \leq e^{-\frac{d_3}{8}} \leq e^{-2\log n} = \frac{1}{n^2}.$$

Finally, by Lemma 4, we get that

$$\Pr(X < \frac{d_3}{2}) \leq \Pr(Y < \frac{d_3}{2}) \leq \frac{1}{n^2},$$

hence

$$\Pr(X \geq \frac{d_3}{2}) \geq 1 - \frac{1}{n^2},$$

which completes the proof. □

Lemma 10. *Let S_1 and S_2 be two disjoint sets of nodes, and $U = S_1 \cup S_2$ be their union. Let $|U| = n$ and $m = \min\{|S_1|, |S_2|\}$, where $|S|$ denotes the size of set S. Let $\mathcal{P} =< u_1, u_2, \ldots, u_n >$ be a permutation of nodes in U, selected uniformly at random from the set of all possible permutations. We mark a node u_i, $1 < i \leq n$, with probability $1 - \frac{1}{n}$ if $u_i \in S_2$, and $u_{i-1} \in S_1$. Let $S_2' \subseteq S_2$ be the set of nodes marked. Then, there exists constants $c_1, c_2 \in O(1)$ such that*

$$\Pr\left(|S_2'| \geq \frac{m}{c_1}\right) \geq 1 - \frac{1}{n},$$

if $m \geq c_2\lceil \log n \rceil$.

Proof: To generate a random permutations of nodes we can do the following. We assign a random point from $[0,1]$ to each node. Then the permutation that orders the points in ascending order according to their x-coordinate determines a random permutation of the nodes. In the remaining of the proof, we use random points in [0,1] to represent and refer to nodes.

We set $c_2 = 72$. Let $d_1 = |S_1|$, and

$$d_2 = d_3 = \lfloor \frac{m}{2} \rfloor.$$

Therefore, $d_1 \geq 72\lceil \log n \rceil$, and $d_2 = \lfloor \frac{m}{2} \rfloor \geq 36\lceil \log n \rceil$. Note that $d_1 \geq 2d_2$. Let

$$I = \left\{ [0, \frac{1}{d_2}), [\frac{1}{d_2}, \frac{2}{d_2}), \ldots, [\frac{d_2 - 1}{d_2}, 1] \right\},$$

and $I' = I$.

If we select d_1 points uniformly at random from $[0, 1]$, by Lemma 9, we get that with probability at least $1 - \frac{1}{n^2}$, there is a set $I'' \subseteq I'$ of size at least $\frac{d_3}{2} \geq 18\lceil \log n \rceil$ such that every interval in I'' contains at least one point from S_1. We use Lemma 9 one more time, this time by setting $I \leftarrow I$, $I' \leftarrow I''$, and selecting $|S_2|$ points from $[0, 1]$. if $|I''| \geq 18\lceil \log n \rceil$ (which the case with probability at least $1 - \frac{1}{n^2}$), then with probability at least $1 - \frac{1}{n^2}$, there is a set $J \subseteq I''$ of size at least $\frac{|I''|}{2}$ such that every interval in J contains a point from S_2. Therefore, since $J \subseteq I''$, and by a union bound, with probability at least $1 - \frac{2}{n^2}$, there is a set J of size at least $\frac{|I''|}{2} \geq \frac{|I|}{4} \geq 9\lceil \log n \rceil$ such that every interval in J contains at least a point from both S_1 and S_2. We mark an interval in J with probability $1 - \frac{1}{n}$ if the interval has two adjacent points p and q such that $p \in S_1$, $q \in S_2$, and $p_x < q_x$, that is the x-coordinate of p is smaller than that of q. Note that every interval in J has points from both S_1 and S_2, and points in that interval are distributed uniformly at random. Therefore, for an interval in J, the probability that there are two points $p \in S_1$, $q \in S_2$ with $p_x < q_x$ is at least $\frac{1}{2}$. Hence an interval in J is marked with probability at least $\frac{1}{2}(1 - \frac{1}{n})$. Let X_i, $1 \leq i \leq |J|$, be $\{0, 1\}$ random variables such that $X_i = 1$ if the ith interval in J is marked. By the above discussion, we have

$$\Pr(X_i = 1 | X_{i-1} = x_{i-1}, \ldots, X_1 = x_1) \geq \frac{1}{2}(1 - \frac{1}{n}).$$

Let Y_i, $1 \leq i \leq |J|$, be i.i.d Bernoulli random variables with

$$\Pr(Y_i = 1) = \frac{1}{2}(1 - \frac{1}{n}).$$

Let $X = \sum_{i=1}^{|J|} X_i$ and $Y = \sum_{i=1}^{|J|} Y_i$. Therefore

$$\mu_y = \mathbb{E}[Y] = \frac{1}{2}(1 - \frac{1}{n})|J|.$$

By a Chernoff bound, we get

$$\Pr\left(Y \leq (1 - \delta)\mu_y\right) \leq e^{-\frac{\delta^2}{2}\mu_y}.$$

Therefore, setting $\delta = \frac{3}{4}$, we get

$$\Pr\left(Y \leq \frac{1}{4}\mu_y\right) \leq e^{-\frac{9}{32}\mu_y},$$

hence, assuming $n \geq 9$ (which is the case as $c_2 = 72$, and $n \geq 2m$), and $|J| \geq 9\lceil \log n \rceil$ (which is the case with probability at least $1 - \frac{2}{n^2}$), we get $\mu_u \geq 4\lfloor \log n \rfloor$, thus

$$\begin{aligned}\Pr\left(Y \leq \frac{1}{4}\mu_y\right) &\leq e^{-\frac{9}{32} \cdot 4\lceil \log n \rceil} \\ &\leq \frac{1}{n^{\frac{9}{8}}}.\end{aligned}$$

Consequently, by Lemma 4, we get

$$\Pr\left(X \leq \frac{1}{4}\mu_y\right) \leq \frac{1}{n^{\frac{9}{8}}},$$

with probability at least $1 - \frac{1}{n^2}$. Adding everything together and using a union bound, we get that there are more than

$$\begin{aligned}\frac{1}{4}\mu_y &\geq \frac{1}{4} \cdot \frac{1}{2}(1 - \frac{1}{n})|J| \\ &\geq \frac{1}{4} \cdot \frac{1}{2}(1 - \frac{1}{n})\frac{|I|}{4} \\ &= \frac{1}{4} \cdot \frac{1}{2}(1 - \frac{1}{n})\frac{\lfloor \frac{m}{2} \rfloor}{4} \\ &= \frac{1}{32}(1 - \frac{1}{n})\lfloor \frac{m}{2} \rfloor\end{aligned} \tag{4}$$

pairs of adjacent points $p \in S_1$ and $q \in S_2$ $p_x < q_x$, with probability at least

$$1 - \left(\frac{2}{n^2} + \frac{1}{n^{\frac{9}{8}}}\right).$$

Since $c_2 = 72$ and that $m \geq c_2 \log n$, we get that $m \geq 72$. Also, by definition of m, we have $n \geq 2m$ thus $n \geq 144$. When $n \geq 144$, we have

$$1 - \left(\frac{2}{n^2} + \frac{1}{n^{\frac{9}{8}}}\right) \geq 1 - \frac{1}{n}.$$

Also, when $n \geq 2m$, and $m \geq 72$, we get

$$\lfloor \frac{m}{2} \rfloor \geq \frac{m - 1}{2} \geq \frac{m}{(2 + \frac{2}{71})},$$

Thus, we have

$$\begin{aligned}\frac{1}{32}(1 - \frac{1}{n})\lfloor \frac{m}{2} \rfloor &\geq \frac{1}{32}(1 - \frac{1}{144})\frac{m}{(2 + \frac{2}{71})} \\ &> \frac{m}{66}.\end{aligned}$$

Therefore, by (4), we get that the number of intervals marked is at least $\frac{m}{c_1}$, with probability at least $1 - \frac{1}{n}$, where $c_1 = 66$. Finally, note that the number of intervals marked is a lower bound of the number of nodes marked. □

Now we are ready to prove Theorem 7.

Proof of Theorem 7: Let $\mathcal{R} = (r_1, r_2, \ldots, r_q)$ be an increasing sequence of q real numbers such that $r_1 = 1$, $n - 1 < r_q \leq n$, and

$$r_{i+1} = \begin{cases} r_i + 1 & \text{if } r_i \leq c_2 \log n \\ (1 + \frac{1}{c_1})r_i & \text{if } c_2 \log n < r_i \leq \frac{n}{2} \\ r_i + \frac{n - r_i}{c_1} & \text{if } \frac{n}{2} < r_i \leq n - c_2 \log n \\ r_i + 1 & \text{if } n - c_2 \log n < r_i \leq n - 1 \end{cases}$$

for $1 \leq i \leq m-1$, where c_1 and c_2 are constants from Lemma 10. Note that $q = O(\log n)$. We say that network is in stage s_i, $1 \leq i \leq q-1$ with respect to λ, if the number of nodes knowing λ is at least r_i and less than r_{i+1}, and is in the final stage s_q if every node knows λ. Therefore, the network is initially in stage s_1 with respect to every vector λ.

Let $X_1, X_2, \ldots$ be $\{0,1\}$-valued random variables such that $X_i = 1$ if at least one of the following conditions holds: i) the network is in stage s_q at the beginning of round i, ii) the network moves to a higher stage by the end of round i compared to the stage it was in at the beginning of round i. Next, we show that

$$Pr(X_i = 1 | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) \geq 1 - \frac{1}{n}. \quad (5)$$

Note that, if we are in stage s_j, $j < q$, then there exists a node that does not know λ, hence we can find a pair of adjacent nodes u_t and u_{t+1} in the permutation such that u_t knows λ, while u_{t+1} does not. By Lemma 1, the probability that u_{t+1} knows λ after receiving a block from u_t is $1 - \frac{1}{n}$. Therefore, for the case where $1 \leq j \leq c_2 \log n$ or $n - c_2 \log n \leq j < q$, we move to a higher stage by the end of round i, with probability $1 - \frac{1}{n}$, hence (5) holds in this case.

Now, consider a second case where $c_2 \log n < j \leq \frac{n}{2}$. Let S_1 be the set of nodes that know λ at the beginning of round i, and S_2 be the set of ones that do not know λ at the beginning of round i. Using S_1 and S_2 in Lemma 10, we get that the number of nodes marked is at least $\frac{r_j}{c_1}$ with probability at least $1 - \frac{1}{n}$. Note that the nodes that are marked represent the nodes that get to know λ in round i. Therefore, by Lemma 10, the number of nodes knowing λ increases by a factor of at least $\frac{1}{c_1}$ (hence, we move to a higher stage), with probability at least $1 - \frac{1}{n}$. Similarly, using Lemma 10, we can show that (5) holds for the remaining case of $\frac{n}{2} < j \leq n - c_2 \log n$.

Let $Y_1, Y_2, \ldots$ be a collection of independent Bernoulli variables with $\Pr(Y_i = 1) = 1 - \frac{1}{n}$. Let

$$X = \sum_{i=1}^{m+k+r} X_i, \text{ and } Y = \sum_{i=1}^{m+k+r} Y_i,$$

where r is a non-negative integer. We have

$$\begin{aligned}
\Pr(Y < m) &\leq \binom{m+k+r}{k+r+1} \frac{1}{n^{k+r+1}} \\
&= \binom{m+k+r}{m-1} \frac{1}{n^{k+r+1}} \\
&\leq \frac{(m+k+r)^{m-1}}{(m-1)! \cdot n^{k+r+1}} \\
&\leq \frac{(m+k+r)^{m-1}}{n^{k+r+1}} \\
&\leq \frac{1}{n^k} e^{(m-1)\log(m+k+r) - (r+1)\log n} \\
&= \frac{1}{n^k} e^{m\log(m+k+r) - r\log n} \cdot e^{-\log(m+k+r) - \log n}.
\end{aligned} \quad (6)$$

Let $A = \frac{m}{\log n}$, $B = m + k$, and

$$r_0 = A \log\left(2(A+B)\log(A+B)\right).$$

Note that $A \geq 1$, $B \geq 1$. By lemma 8, we have

$$r \geq A \log(B + r), \ \forall \ r \geq r_0.$$

Thus, replacing A and B with $\frac{m}{\log n}$ and $m + k$, respectively, we obtain

$$r \log n \geq m \log(m + k + r), \ \forall \ r \geq r_0.$$

Therefore, by (6), we have

$$\begin{aligned}
\Pr(Y < m) &\leq \frac{1}{n^k} e^{m\log(m+k+r) - r\log n} \cdot e^{-\log(m+k+r) - \log n} \\
&\leq \frac{1}{n^k} e^{-\log(m+k+r) - \log n},
\end{aligned} \quad (7)$$

when $r \geq r_0$. Since $m = O(\log n)$, we get

$$r_0 = O(\log k + \log\log n).$$

By lemma 4, we have

$$\Pr(X \geq d) \geq \Pr(Y \geq d)$$

for every non-negative integer d. Hence, we have

$$\Pr(X < m) \leq \Pr(Y < m).$$

Thus, by (7), we get

$$\Pr(X < m) \leq \frac{1}{n^k} e^{-\log(m+k+r) - \log n}.$$

That is, the probability that every node knows λ in $m + k + r_0$ rounds is at least

$$1 - \frac{1}{n^k} e^{-\log(m+k+r) - \log n}.$$

There are n^k vectors λ in total. By a union bound, all nodes know about every vector λ in $m + k + r_0$ rounds with probability at least

$$\begin{aligned}
1 - \frac{1}{n^k} e^{-\log(m+k+r_0) - \log n} \cdot n^k &= 1 - \frac{1}{n(m+k+r_0)} \\
&\geq 1 - \frac{1}{nk},
\end{aligned} \quad (8)$$

which proves the theorem. □

7 Simulations

In this section, we resort to simulations to verify our theoretical results and evaluate the coding benefits as well as its computation overhead. We wrote a C program that can perform coding operations on randomly generated real data blocks of any predefined size. We also propose a practical 3-step pipeline technique to solve the problem of coding latencies and discuss the optimization of block division given the bulk data to be transferred.

We simulate RLNC with random permutation as well as a non-coding baseline dissemination protocol called *Random Block* with random permutation described as follows. Let S_v^t be the set of blocks node v possesses at

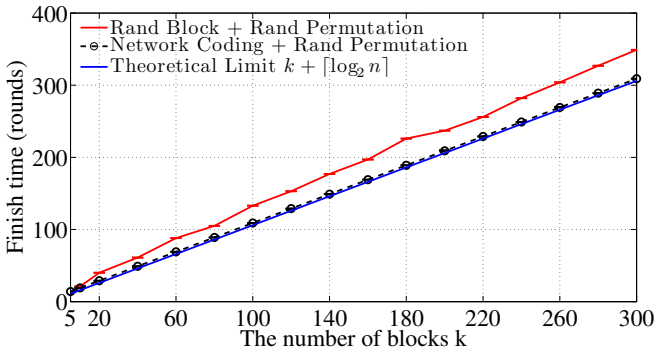


Fig. 3. The finish time as a function of k with error bars. Each point represents the average finish times of at least 10 runs. $n = 60$.

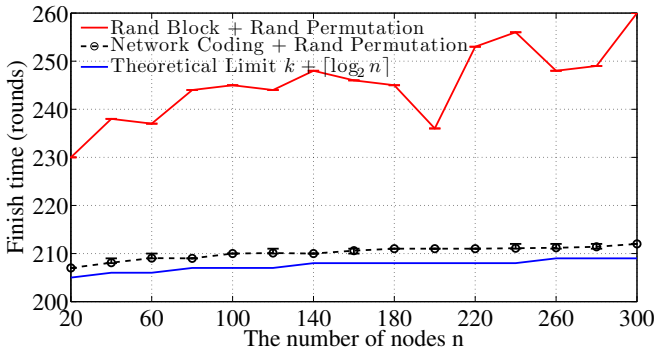


Fig. 4. The finish time as a function of n with error bars. Each point represents the average finish times of at least 10 runs. $k = 200$.

the beginning of round t. In round t, if v is sending to u, it selects a block uniformly at random from the difference set $S_v^t \backslash S_u^t$ to send to u. Note that $S_v^t \backslash S_u^t$ may be empty, in which case v cannot help u. This method requires all the nodes to exchange their buffer maps periodically and is similar to the swarming idea in BitTorrent P2P networks.

For RLNC, we use a field size of $q = 2^8$, and apply progressive decoding, that is, whenever a node receives a new (coded) block, it will perform Gaussian elimination for this block together with all previously received blocks. Such progressive decoding will allow each node to pipeline the entire 3-step process of receiving a block, decoding it and encoding a new block, and sending out a new block.

7.1 Verifying Theoretical Results

First, we fix the number of nodes to 60, which is the usual number of servers in a computer cluster such as Hadoop or Spark. We vary k, and for each k perform 10 independent broadcast (one-to-all) experiments to record the finish rounds. Fig. 3 plots the results of different algorithms with error bars specifying the minimum and maximum finish rounds for the corresponding k. We observe that RLNC with random permutation can closely approach the theoretical limit $k + \lceil \log_2 n \rceil$ rounds, even for a small k, with *extremely short error bars*, confirming that our finisth time bound $k + O(\log n + \log k)$ is tight. In fact, only when $k = 200$ or $k = 260$, the finish times of RLNC in 10 different runs have varied for only 1 round. This implies that our randomized algorithm can *almost surely* finish in just a little above $k + \lceil \log_2 n \rceil$ rounds and that absolutely optimal (not only order-optimal) block pipelining in k is achieved. On the other hand, Random Block, although knowing the buffer states of other nodes, is apparently worse than RLNC, and finishes in about $1.3k - 1.4k$ rounds as k scales.

Second, we fix the number of blocks to $k = 200$. For example, in the logistic regression example of Sec. 3, if 200MB are to be broadcast and each block has a size of 1MB, then $k = 200$. We vary n, and for each n perform 10 independent broadcast experiments. The finish times are plotted in Fig. 4 with error bars. Again, the error bars are so short that for random block, there is no variation across different runs, and for network coding, the variation of finish times is at most 1 round for each n. In Fig. 4, the finish time of RLNC is at most 4 rounds more than the theoretical limit $k + \lceil \log_2 n \rceil$ in all the runs for all n. In contrast, the finish time of random block could be $30 - 50$ rounds more than $k + \lceil \log_2 n \rceil$, and such a gap increases in trend as n grows.

7.2 Coding Latency and Pipelining

Next, we analyze whether and how the computation overhead of network coding can affect the dissemination. We run our program for a given set of n, k and block size on a single machine with a 2.6 GHz Intel Core i7 processor and record the elapsed time of the entire program for RLNC until broadcast completes. Since no real network transfer happened, we can divide the total elapsed time by the number of nodes n to estimate how much time on average each node will spend on encoding and decoding operations in a parallel cluster of n nodes. Fig. 5 plots the computation overhead of each node for the entire broadcast session. We find *1)* that the computation time per node is linear to the block size, *2)* that is a convex function of the number of blocks k, and *3)* that the number of nodes does not affect computation time *per node*.

The above observations not only allow us to estimate the computation overhead under a certain parameter setting, but also enables us to carefully divide the blocks so that the coding latency can be hided when the process of receiving a block, decoding it and encoding a new block, and sending out the new block is fully pipelined at each node in the cluster.

Note that at most 1 block is sent in each round by each node according to the protocol. Without coding, the length of each round is just the time to transfer one block, which is *block size/link bandwidth*. When coding is used, at most 1 block is received/decoded/encoded/sent in each round by each node. In each round, each node can do the following 3 steps simultaneously in parallel:

1) receiving a new coded block;
2) decoding the block received in the previous round and encoding a new block;
3) sending out a new block encoded in the previous round.

If the time spent on decoding and encoding per round by each node is smaller than the time to transfer (send or receive) a block, coding operations will not become the bottleneck of the pipelined process at each node, and the length of each round is still *block size/link bandwidth*. However, if the time spent on coding per round per node is greater than the time to transfer a block, coding will be the bottleneck, and the length of each round will be the time to decode a block plus the time to encode a block. Therefore, we should carefully choose a block dividing strategy to leverage the throughput benefit of network coding while hiding its coding latency.

7.3 Optimized Block Division

We illustrate our block dividing strategy with a case study of broadcasting 300 MB of data in a cluster of $n = 60$ nodes. Fig. 6(a) plots the estimated decode+encode time as well as transfer time per round (*block size/link bandwidth*) spent by each node when the 300 MB data is divided into k blocks. The decode+encode time can be estimated from the existing measurements from Fig. 5. For example, if $k = 50$, the block size is $300/50 = 6$ MB. We first obtain from the profile of Fig. 5(b) that the computation time per node for $k = 50$ with a block size of 10 KB is 0.38 s. Then, due to the linearity of computation time in block size (as have been discussed above for Fig. 5(a)), the computation time per node for a block size 6 MB is 0.38×6 MB $/10$ KB $= 233.47$ s. Furthermore, since network coding finishes broadcasting $k = 50$ blocks in 58 parallel rounds ($58 = 50 + \lceil\log_2 60\rceil$+2), the computation time per round at each node is $233.47/58 = 4$ s, which is smaller than the transfer time per round at each node 6 MB/10 Mbps $= 4.8$ s, assuming 10 Mbps link bandwidth. Thus, coding is not the bottleneck of the system, and the total time to finish broadcast is 4.8 s $\times58$ rounds $= 278.4$ s. If coding becomes the bottleneck, the total time to finish broadcast would be the computation time per round at each node times the total number of rounds needed. As such, Fig. 6(b) plots the estimated total broadcast finish time (seconds) of RLNC as k varies, under different available network bandwidth.

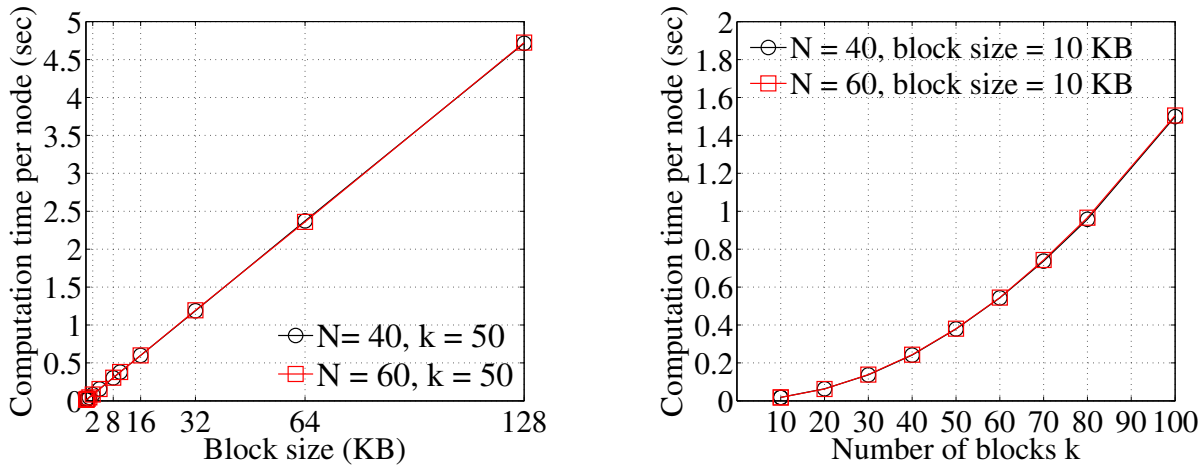
Fig. 6(a) shows that as k becomes larger, the computation time *per round* per node will eventually exceed the transfer time per round per node and become a bottleneck. On the other hand, when k is smaller, the throughput could be lower, since the finish round of RLNC is $k + \lceil\log_2 n\rceil + C$, where C is $2 \sim 4$, and the download throughput at each node is $k/(k+\lceil\log_2 n\rceil+C)$, which approaches 1 as k increases. As k varies, there is clearly a tradeoff between throughput efficiency and coding complexity. Therefore, a good idea is to set k to *the largest value at which computation is not a bottleneck*, or equivalently, to *the value such that computation time equals to transfer time per round per node* in Fig. 6(a). We observe that such critical value of k increases as the network bandwidth decreases. For 20 Mbps links, k should be at most 30. For 10 Mbps links, k should be at most 60, and for 100 Mbps, links, k could be greater than 100. By further checking Fig. 6(b), we find that such critical k's (30, 60) indeed achieve the minimum total broadcast finish times under 20 Mbps and 10 Mbps links, respectively, justifying our strategy of block dividing. It is worth noting that when network bandwidth is *extremely large*, coding time will inevitably become a bottleneck, even for a small k. In such cases, RLNC may not be useful, since the network is already fast.

Finally, the bulk data size does not affect the choice of the optimal k. In the previous 300 MB data example, if the data becomes 600 MB while k remains the same, both computation time and transfer time per round will double at each node. Thus, the critical k such that transfer latency equals to computation latency remains the same. For this reason, disseminating an extremely large data bulk in k blocks is the same as dividing the bulk into several smaller batches, each comprised of k blocks, and disseminating batches *one after another*. However, the more fine-grained batch-based approach will better utilize the pipeline: when the first batch is close to finish and few nodes can receive new blocks, the second batch can start to better utilize the resources.

8 Concluding Remarks

In this paper, we have bounded the finish time of disseminating k blocks to n nodes, using random linear network coding. We consider a simple node permutation rule, in which each node transmits a block to its successor in the permutation in each round. We show that with arbitrary permutation, the finish time is at most $k + n$ rounds if the field size q for coding is large enough. With random permutation, the finish time is $k + O(\log n + \log k)$, which is very close to the best possible theoretical limit $k + \lceil\log_2 n\rceil$. Simulation results show that our simple algorithms can achieve absolutely optimal pipelining in k (not only order-optimal) and finishes broadcasting k blocks to all n nodes in no more than $k + \lceil\log_2 n\rceil + 4$ rounds in all our experiments with up to 300 nodes and up to 300 blocks.

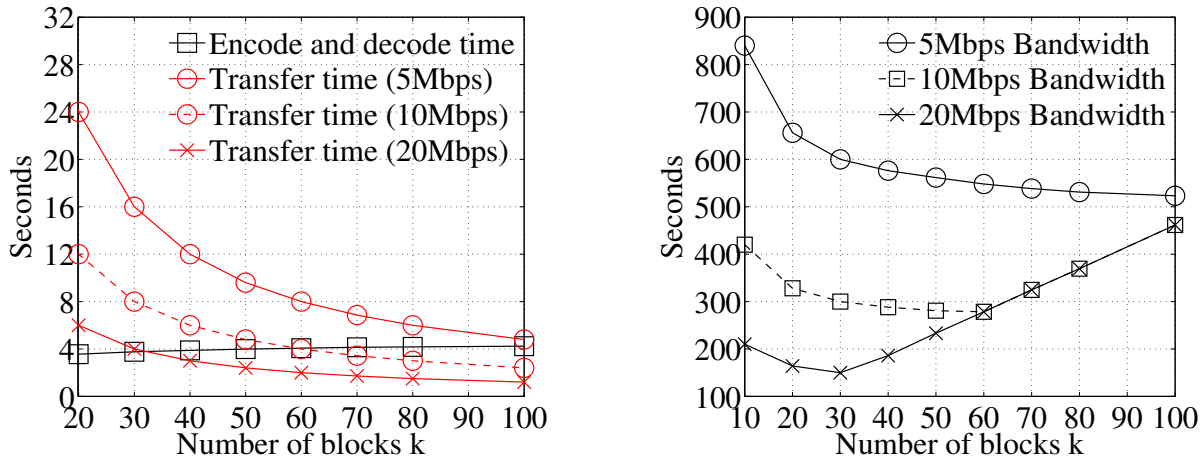
We demonstrate the potential application of the proposed algorithms to content distribution in computing clusters. We carefully evaluate the computation latency of RLNC, and propose practical pipelining schemes to hide such latencies and optimize the total data transfer time considering both network and computing configurations. To strike a balance between throughput and computing efficiencies, our simulation results show that with a 2.6 GHz Intel Core i7 processor, it is best to divide the data into 60 blocks if the available network bandwidth is 10 Mbps, while in a 20 Mbps network, it is best to divide the data into only 30 blocks, regardless of the size of the bulk data to be transferred.



(a) Computation time vs. block size

(b) Computation time vs. k

Fig. 5. Mean encode + decode time per node for the entire broadcast session.



(a) Computation and transfer time spent per node per round

(b) Total Broadcast Finish Time

Fig. 6. a) Estimation of the time spent on computation and transfer per round by each node, and b) Estimation of the total broadcast finish time (seconds) under different network bandwidth and different block numbers k. As k varies, there is a tradeoff between throughput and computation efficiencies.

REFERENCES

[1] A. Bar-Noy and S. Kipnis, "Broadcasting Multiple Messages in Simultaneous Send/Receive Systems," *Discrete Applied Mathematics*, vol. 55, pp. 95–105, 1994.

[2] S. Sanghavi, B. Hajek, and L. Massoulie, "Gossiping with Multiple Messages," in *Proc. IEEE INFOCOM*, Anchorage, Alaska, 2007.

[3] S. Deb, M. Médard, and C. Choute, "Algebraic Gossip: A Network Coding Approach to Optimal Multiple Rumor Mongering," *IEEE Trans. Inform. Theory*, vol. 52, no. 6, pp. 2486–2507, June 2006.

[4] B. Haeupler, "Analyzing network coding gossip made easy," in *Proc. of ACM STOC*, 2011.

[5] D. Niu and B. Li, "Asymptotic optimality of randomized peer-to-peer broadcast with network coding," in *Proc. of IEEE INFOCOM*, 2011.

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. of OSDI*, 2004.

[7] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. of the 2nd USENIX Conference on Hot topics in Cloud Computing*, 2010.

[8] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. of EuroSys*, 2007.

[9] M. Chowdhuryand, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *Proc. of ACM SIGCOMM*, 2011.

[10] D. M. Chiu, R. W. Yeung, J. Huang, and B. Fan, "Can Network Coding Help in P2P Networks?" in *Proc. International Workshop on Network Coding*, Boston, April 2006.

[11] J. Li, P. A. Chou, and C. Zhang, "Mutualcast: An Efficient Mechanism for Content Distribution in a Peer-to-Peer (P2P) Network," in *Proc. of ACM SIGCOMM Asia Workshop*, Beijing, China, April 2005.

[12] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou, "Utility Maximization in Peer-to-Peer Systems," in *Proc. ACM SIGMETRICS*, Annapolis, Maryland, USA, June 2008.

[13] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized Decentralized Broadcasting Algorithms," in *Proc. IEEE INFOCOM*, Anchorage, Alaska, USA, May 2007.

[14] C. Feng, B. Li, and B. Li, "Understanding the Performance Gap between Pull-based Mesh Streaming Protocols and Fundamental Limits," in *Proc. IEEE INFOCOM*, Rio de Janeiro, Brazil, April 19-25 2009.

[15] R. Kumar, Y. Liu, and K. Ross, "Stochastic Fluid Theory for P2P Streaming Systems," in *Proc. IEEE INFOCOM*, Anchorage, Alaska, USA, 2007.
[16] J. Mundinger, R. Weber, and G. Weiss, "Optimal Scheduling of Peer- to-Peer File Dissemination," *Journal of Scheduling*, 2007.
[17] M. Al-Fares, S. Radhakrishnan, B.Raghavan, N.Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. of NSDI*, 2010.
[18] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, "Design and evaluation of a real-time url spam filtering service," in *Proc. of IEEE Symposium on Security and Privacy*, 2011.
[19] T. Hastie, R. Tibshirani, and J. Friedman., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY: Springer, 2009.
[20] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, "Large-scale parallel collaborative filtering for the netflix prize," in *Proc. of AAIM*. Springer-Verlag, 2008, pp. 337–348.
[21] P. A. Chou, Y. Wu, and K. Jain, "Practical Network Coding," in *Proc. of the 41st Annual Allerton Conference on Communication, Control and Computing*, October 2003.
[22] M. Wang and B. Li, "R^2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming," *IEEE J. on Sel. Areas in Communications*, December 2007.
[23] M. Khabbazian, F. Kuhn, N. Lynch, M. Medard, and A. ParandehGheibi, "Mac design for analog network coding," in *Proc. of FOMC*, 2011.
[24] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proc. of IEEE INFOCOM*, 2005.

Majid Khabbazian. Majid Khabbazian received his undergraduate degree in computer engineering from Sharif University of Technology, Iran, and his Masters and Ph.D. degrees both in electrical and computer engineering from the University of Victoria and the University of British Columbia, Canada, respectively. From 2009 to 2010, he was a postdoctoral fellow at the Computer Science And Artificial Intelligence Lab, MIT. Currently, Dr. Khabbazian is an assistant professor in the Department of Electrical and Computer Engineering, University of Alberta, Canada. His research interest is mainly in designing efficient, robust and secure distributed algorithms. Dr. Khabbazian is a member of IEEE and ACM.

Di Niu. Di Niu received the B.Engr. degree from the Department of Electronics and Communications Engineering, Sun Yat-sen University, China, in 2005 and the M.A.Sc. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, in 2009 and 2013. Since September, 2012, he has been with the Department of Electrical and Computer Engineering at the University of Alberta, where he is currently an Assistant Professor. His research interests span the areas of multimedia delivery systems, cloud computing and storage, data mining and statistical machine learning for social and economic computing, distributed and parallel computing, and network coding. Dr. Niu is a member of IEEE and ACM.