

# A SURVEY OF RANDOMIZED NETWORK CODING

STEVEN CHENG

## 1. INTRODUCTION

Optimizing multicast throughput has been a classical problem in network information flow. Unicast throughput is bounded by the max-flow of the network. By applying network coding, we can achieve the unicast bound for every receiver in a multicast setting. In addition, computing the required coding scheme to achieve this bound can be done in polynomial time, as opposed to finding an optimal multicast routing scheme which is NP-hard [3].

Thus network coding seems to be an obvious solution where multicast is needed. However, even the polynomial time algorithms are somewhat expensive, and require extra coordination between nodes in the network. It turns out, however, that a randomly generated code has a fairly high probability of being a solution to a multicast problem.

## 2. OVERVIEW

In order to do network coding, we need a method of combining symbols. The immediate answer is simply bitwise XOR. This can be applied in very simple situations, such as the classic butterfly network example. In general, we need something a little more powerful, linear codes.

Linear codes are simply linear functions, but since we are dealing with discrete data, we base our calculations in a finite field rather than the field of real numbers. In particular, Galois fields are used in practice. This defines an XOR as the addition operation, and uses polynomial multiplication to define the multiplication operation. Though it is more complicated than regular multiplication, Galois field multiplication can be implemented efficiently with lookup tables.

Of course, we are still just doing linear algebra, so you can generally think of this as an implementation detail. General intuition from linear algebra on  $\mathbb{R}^n$  usually suffices to get the feeling of most results.

We start with the basic results of network coding, which state are that linear codes suffice to achieve the max-flow bandwidth to every receiver in a graph, given sufficient field size. This has been proved in [1] and in [14] using different frameworks.

Most deterministic network coding algorithms must go through a phase where a coding scheme is globally computed, then information is transmitted along these schemes. This however, is sensitive to changes in the network topology, faulty links, and nodes that leave the system. In order to manage this, these schemes require a fair amount of central control or coordination between nodes.

One of the first randomized network coding algorithms was proposed in [18] to create a network coding scheme for a multicast problem in randomized polynomial time. However, this scheme was still quite centralized, with the randomization being

used mainly to simplify the generation of codes rather than reduce coordination and central control.

Eventually, a new approach, which we will call distributed randomized network coding, was created.[4] gives implementation details for this approach which was first proposed in [12]. This novel scheme simply requires the coding scheme to be distributed with the packet, achieving greater independence between nodes in exchange for a little bit of overhead. It was later realized that this scheme could be used in a p2p network setting, rather than just for multicast. We will begin our survey here since the idea behind the distributed randomized network coding approach is more easily seen and analyzed in this setting.

### 3. P2P NETWORKS

**3.1. Models and Notation for P2P networks.** We use the following model and notations, which are mostly the same as that in [6]:

- (1) the network is modeled as a directed graph  $G = (V, E)$ . We will not consider any notion of weights on the edges of the graph and simply assume that transmitting over an edge takes 1 time unit.
- (2) The field over which we are coding is  $F_q$ , usually the galois field  $GF(2^{\lg q})$ .
- (3) We consider data split into fixed size blocks, sequences of symbols from the field  $F_q^k = B$ . Here  $B$  is the set of all possible blocks.
- (4) The collection of data is a fixed vector  $M = (m_1, m_2, m_3, \dots, m_k) \in B^k$ , that we either wish to store or distribute. We can easily turn any sequence of bytes to and from this form, possibly with some extra padding.
- (5) Each node  $v \in V$  has a set  $D_v \subset F_q^n \times B$  of (coding vector, coded block) pairs, often called packets. In a trusted environment, we can assume that for every  $(cv, cb)$  pair is such that  $cb = \sum_{i=1}^n cv[i] \cdot m_i$ . In this case, we can see that this is a set of linear equations on the  $n$  blocks. If the set is of size  $n$  and the vectors are linearly independent, then we can solve for the original blocks by doing Gaussian elimination. When the environment is not trusted, we must find ways to validate the data or correct for malicious data.
- (6) We also have for each node a vector space  $S_v = span\{cv : (cv, cb) \in D_v\}$
- (7) Each node can generate a new valid (coding vector, coded block) pair by linearly combining the ones it has. This will be done using random coefficients.

All the applications use randomized network coding, which means that the packets they exchange are combinations of coding vectors and coded data. Each node stores a linearly independent set of these vectors. Once a large enough set of linearly independent vectors is collected, we then have a set of linear equations that can be solved by Gaussian elimination or another matrix inversion algorithm, which will give us the original sequence  $M$ . Since we are generally considering trusted environments, the analysis is done using properties of the coding vectors, rather than the data itself.

**3.2. Analysis of Random Network coding for P2P networks.** We first need some definitions:

- A vector  $g$  is helpful to a node  $v$  if  $g \notin S_v$ .

- A node  $u$  is helpful to a node  $v$  if  $\exists g \in S_u$  so that  $g \notin S_v$ , or equivalently if  $S_u \not\subseteq S_v$ .

Now we state the key result from [6] that allows us to analyze randomized network coding applications, and provides justification for its use.

**Lemma 3.1.** *Let  $S_u$  and  $S_v$  be subspaces of  $F_q^k$ , with  $S_u \not\subseteq S_v$  if  $g$  is a randomly generated vector in  $S_u$  then  $Pr(g \notin S_v) \geq 1 - \frac{1}{q}$ . A picture of this situation is:*

$$u \xrightarrow{g} v$$

*Proof.* Since  $S_u \not\subseteq S_v$ , let  $x \in S_u - S_v \neq \emptyset$ .

There exists a basis of  $S_u$ ,  $B = \{x, b_1, b_2, \dots, b_{\dim(S_u)-1}\}$ .

Any random non-zero linear combination of the vectors in  $S_u$  has a unique representation in  $B$ ,  $\beta_0 x + \sum_{i=1}^{\dim(S_u)-1} \beta_i b_i$ .

If  $x$  has a non-zero coefficient, then the linear combination is not in  $S_v$ , so:

$$Pr(g \notin S_v) \geq Pr(\beta_0 \neq 0) = 1 - \frac{1}{q} \quad \square$$

This lemma is the probability the node  $v$  will help  $u$ , given that it can help  $u$ . In other words, the probability that the randomly generated vector  $g$  is helpful to  $v$ . This lemma tells us that we can make this probability arbitrarily small by increasing the field size  $q$ .

**Corollary 3.2.** *Let  $S_u, S_v, S_w$  be subspaces of  $F_q^k$  such that  $S_u \not\subseteq S_v$  and  $S_w \subseteq S_u$  then if  $g$  is a randomly generated vector in  $S_u$  then  $Pr(\text{span}\{S_w \cup \{g\}\} \not\subseteq S_v) \geq 1 - \frac{1}{q}$ . A picture of this situation is:*

$$u \xrightarrow[t]{g} v \xrightarrow[t+1]{g'} w$$

*Proof.* This is a straightforward application of the above lemma. The lower bound on the probability of  $g$  being helpful to  $v$  is the same as  $g$  being helpful to  $w$ .

$$\text{Since } S_w \subseteq S_u, Pr(\text{span}\{S_w \cup \{g\}\} \not\subseteq S_v) = Pr(g \notin S_v) \geq 1 - \frac{1}{q} \quad \square$$

This lemma relates to the probability that, given that that  $w$  is not helpful to  $u$ , and  $v$  is helpful to  $u$ , that  $w$  will be helpful to  $u$  after it receives a helpful vector from  $v$ . This will be useful when we need to consider how the system evolves over time.

**Definition 3.3.** Stochastic ordering [16, 15]:

For real-valued random variables,  $X, Y$ :

$X \prec_{st} Y$  ( $Y$  stochastically dominates  $X$ ) if  $\forall a, Pr(X \geq a) \leq Pr(Y \geq a)$

We will use this definition combined with the following fact to obtain our bounds.

**Fact 3.4.** [15] *If  $X$  and  $Y$  are random variables with finite expectation and  $X \prec_{st} Y$  then  $E[X] \leq E[Y]$ .*

**3.3. Gossip based information dissemination.** Gossip or epidemic based systems have had a lot of development time since their first introduction in [7]. They aim to provide a more distributed way of spreading information. This common theme is shared with distributed randomized network coding, and so it is natural to consider the applying it to gossip based information distribution.

We start out by introducing the basic notion of the algorithm. We assume the system is made up of processors numbered 1 to  $n$ , and there are  $k$  messages, spread

out between them. These processors are connected in a complete network, so that every processor has a direct link to another. Execution proceeds in rounds, every round, each processor will execute pull on one other processor.

We start by giving the algorithm for the network coding case:

---

**Algorithm 1** RandomLinearPull(i)
 

---

Constants: *neighbors*  
 Local Variables:  $D_v, S_v$   
**while**  $\dim(S_i) < k$  **do**  
    $pullProcessor \leftarrow random(1, \dots, n/i)$   
    $(cv, cb) \leftarrow pull(pullProcessor)$   
   **if**  $cv \notin S_i$  **then**  
      $D \leftarrow D \cup (cv, cb)$   
   **end if**  
**end while**

---

Where pull returns a random combination of the packets in  $D_{pullProcessor}$ .  
 We contrast this to the random message selection protocol shown here:

---

**Algorithm 2** RandomSelectPull(i)
 

---

Constants: *neighbors*  
 Local Variable: set of messages  $S_i$   
**while**  $|S_i| < k$  **do**  
    $pullProcessor \leftarrow random(1, \dots, n/i)$   
    $m \leftarrow pull(pullProcessor)$   
   **if**  $m \notin S_i$  **then**  
      $S_i \leftarrow S_i \cup m$   
   **end if**  
**end while**

---

In this case pull returns a random message in  $D_{pullProcessor}$ .

We then follow [6, 12] in considering the situation. We first need a way to analyze the local interaction between two nodes, and for successive interactions between nodes. For this we present their lemma 2.1 in two lemmas:

We will analyze the system by bounding the processes by random variables that are easier to handle. To do this we introduce a few distributions, again following [6].

- Geom(p): random variable with geometric distribution and parameter p
- Bin(N,p): random variable with binomial distribution with parameters N and p
- Random variable with discrete phase distribution [6] T:
  - (1)  $T = inf\{k : X_k = \mathcal{A}\}$ , where  $X_k$  is a discrete Markov Chain on the space  $\mathcal{A} \cup \mathbb{Z}^+$  with:
    - (2)  $X_0 = 1$ , absorption state  $\mathcal{A}$ .
    - (3) Transition probability matrix P is such that  $p_{s,r} = 0$  if  $r \neq s, r \neq s+1$
    - (4)  $p_{s,\mathcal{A}} > 0$
    - (5)  $Pr(\lim_{k \rightarrow \infty} X_k = \mathcal{A}) = 1$

Now we are ready to begin the analysis. Here we only present the “pull” results from [6], the paper also contains analysis for “push” based models. The difference is that in a push based model, the processor chooses which node to send a packet to, rather than being polled for a packet. The two models perform very similarly for both network coded and uncoded approaches.

Following the analysis in [6], we split the time into 3 phases based on the dimension of a node  $i = \dim(S_u(t))$ . It turns out that we can only show the gains of random coding in the second phase. The three phases are:

- $i \in I_1 = [1, \frac{k}{2}]$
- $i \in I_2 = (\frac{k}{2}, k - 14\sqrt{k \ln(k)})$
- $i \in I_3 = (k - 14\sqrt{k \ln(k)}, k]$

In the original paper the stochastic bounds and time bounds were split into separate parts for better readability. However, in the interest of space, we will include the stochastic bound proofs in the time bound proofs. Notation is as follows:

- The set of nodes that are unhelpful to u:  $E_u(t) = \{v | S_v(t) \subseteq S_u(t)\}$
- The fraction of nodes that are unhelpful to u:  $h_u(t) = |E_u(t)|/n$
- $T_i$  the number of rounds in which a node has dimension  $i$ 
  - Once  $i = k$  the node can decode all of the original messages.

**Lemma 3.5.** For  $i \in I_1 = [1, \frac{k}{2}]$

- (1)  $E[T_i] \leq \frac{2q}{q-1}$
- (2)  $\sum_{i \in I_1} T_i \leq \frac{q}{q-1}(k + 4 \ln(n) + 2\sqrt{k \ln(n)})$  w.p  $1 - O(\frac{1}{n^2})$

*Proof.* First we obtain the stochastic bound by considering  $E_u(t)$ . Since  $\dim(S_u(t)) = i$ , we can find  $k - i$  uncoded messages  $\{m_{l_1}, m_{l_2}, \dots, m_{l_{k-i}}\}$  such that each one is helpful to u. Clearly,  $E_u(t)$  cannot contain any node that started with one of these messages, so  $E_u(t) \subseteq \{v \in V : v \text{ does not have } m_{l_j} \text{ at time } 0\} = L$  and since the messages are evenly distributed at time 0, we have  $|E_u(t)| \leq |L| = \frac{ni}{k}$ . From this worst case assumption we also have  $h_u(t) \leq \frac{i}{k}$ .

Let the called node be  $v$ , from this we apply the lemma 3.1 to get:

$$\begin{aligned} Pr(S_u(t+1) \neq S_u(t)) &= Pr(v \notin E_u(t))Pr(S_u(t+1) \neq S_u(t) | v \notin E_u(t)) \\ &\geq (1 - \frac{|E_u(t)|}{n})(1 - \frac{1}{q}) \\ &\geq (1 - \frac{ni}{nk})(1 - \frac{1}{q}) \\ &\geq \frac{1}{2}(1 - \frac{1}{q}) \end{aligned}$$

From which it follows that :

$$T_i \prec_{st} Y_i \sim Geom(\frac{1}{2} \left(1 - \frac{1}{q}\right))$$

By applying a Chernoff bound to the geometric random variable, we obtain 1. and 2.  $\square$

This simple analysis works well for this phase, but we need a more complex proof for the following phases.

**Lemma 3.6.** For  $i \in I_2 = (k/2, k - 14\sqrt{k \ln(k)})$

- (1)  $E[T_i] \leq 1 + \log(\ln(2)) + \frac{2q}{q-1} - \log \ln \frac{k'}{i} + O(\frac{1}{n^2})$

$$(2) \sum_{i \in I_2} T_i \leq \frac{k}{2} \left\langle 1 + \log(\ln 2) + \frac{1 + \ln(2)}{\ln(2)} + \frac{2q}{q-1} \right\rangle + O(\ln(n)) + O(\sqrt{k \ln n}) \text{ w.p. } 1 - O\left(\frac{1}{n^2}\right)$$

Proof sketch: First we obtain the stochastic bound  $T_i \prec_{st} Y_i$  where  $Y_i \sim T$  is a discrete phase type distribution as described earlier. We further specify  $Y_i$  to be on the finite state space  $\{1, 2, 3, \dots, \bar{s}_i\}$  and we have the following transition structure  $P$ , which we will use to bound  $T_i$  as we introduce them:

$$p_{s,\mathcal{A}} = (1 - \bar{h}_s) \left(1 - \frac{1}{q}\right), \quad s \leq \bar{s}_i$$

The absorption state is reached when the node increases its dimension (receives a helpful packet). Note that  $Y_i$  is measuring the time in terms of transitions, that it takes to reach this absorption state. We get the  $(1 - \frac{1}{q})$  from 3.1, then we multiply it by the probability that the node contacted can actually produce a helpful packet, contained in the rather complicated  $\bar{h}_s$  variable.

So,  $s$ , the state, keeps track of the state of the rest of the nodes in the system. From this variable we can determine the number of helpful nodes using  $h_s$ . The state  $\bar{s}_i$  is the state where all nodes are helpful.

$$p_{\bar{s}_i,\mathcal{A}} = 1 - p_{\bar{s}_i,\bar{s}_i} = \frac{1}{2} \left(1 - \frac{1}{q}\right)$$

So the lower bound on the probability of getting a helpful packet is the same regardless of which node is contacted, and we get the familiar  $1 - \frac{1}{q}$  from 3.1 again. Lastly, we need to keep track of how the state of the other nodes changes when we do not get a useful message.

$$p_{s,s+1} = \left(1 - \frac{1}{n^3}\right) (1 - p_{s,\mathcal{A}}), \quad s \leq i$$

This gives the probability that more nodes become helpful. Since we only track this if the node has not received a helpful message, we have  $1 - p_{s,\mathcal{A}}$ , multiplied by the probability that more nodes become helpful. This turns out to be the most complicated part of the proof, and this is where the “magic” term of  $14\sqrt{k \ln(k)}$  shows up. After the end of this phase, we move to a different strategy for analysis.

Finally, after all this, the proof goes through an application of Chernoff bounds to get the desired results.

**Lemma 3.7.** For  $i \in I_3 = (k - 14\sqrt{k \ln(k)}, k]$

- (1)  $E[T_i] = O(\ln(n))$
- (2)  $\sum_{i \in I_3} T_i = O\left(\ln(n)\sqrt{k \ln(k)}\right)$  w.p.  $1 - O\left(\frac{1}{n^2}\right)$

The proof of this lemma bounds  $T_i$  by a sum of geometric random variables, after which Chernoff bounds are applied to give the desired results.

This leads us to the main result:

**Theorem 3.8.** For an random linear code based approach with “pull”:

- (1)  $\sum_i^k E[T_i] \leq 3.46k + O(\ln(n)\sqrt{k \ln(k)})$ ,
- (2)  $Pr(T_{RLC} \leq 3.45k + O(\ln(n)\sqrt{k \ln(k)})) = 1 - O\left(\frac{1}{n}\right)$

This theorem is proved by combining the above lemmas.

3.3.1. *Analysis of uncoded approach.* The basic model is simple, rather than nodes having coding vectors and coded blocks, they have sets  $M_v \subseteq M$ . Each node selects a random piece to transmit.

From the paper, we model  $Pr(e = m) = \frac{[m \in M_v]}{|M_v|}$  where  $[p]$  is 1 if  $p$  is true, 0 otherwise, which is a slightly different notation than the paper. Since we are not too concerned with exact performance, and only want a lower bound in order to compare with the random linear coding scheme, we only need to consider the phase  $i \in (k/2, k]$  where the uncoded approach takes the most of its time. The difference in this analysis is that  $i$  refers to the minimum number of messages received by each node.

**Lemma 3.9.** For  $i \in (\frac{k}{2}, k] : T_i \succ_{st} Geom(\frac{k-i}{i})$

The idea behind this result is that if all nodes have at least  $i$  messages, then the probability of this minimum number increasing is  $\frac{k-i}{i}$ , which is the number of messages that could help the minimum, divided by the size of the minimum. Then the geometric distribution measures how long it takes to get that success.

**Theorem 3.10.** For a random message selection approach with pull and  $k = \alpha n$ :

- (1)  $E[T_{RMS}] = \Omega(n \ln(n))$
- (2)  $\lim_{k \rightarrow \infty} Pr(T_{RMS} = \Omega(n \ln(n))) = 1$

The proof follows by summing the results of the previous lemma, we do not need to consider the phase  $i \in [1, k/2)$  to prove this lower bound.

3.3.2. *Remarks.* We compare the two major results in a table:

	LinearCode	MessageSelect
$E[T]$	$3.46k + O(\ln(n)\sqrt{k \ln(k)})$	$\Omega(n \ln(n))$
$Pr(T \leq E[T])$	$1 - O(\frac{1}{n})$	1 as $k \rightarrow \infty$

We can see that the performance of message selection is lower bounded by the number of processors in the system, while the linear codes are upper bounded by the logarithm of the number of processors in the system. As the ratio  $\frac{n}{k}$  increases, the advantage of network coding also increases. Here, the topology of the network may have some effect, the network coding advantage may be much greater if different network topologies are used.

Since this paper is mainly focused on theoretical analysis, many interesting protocol details are omitted. The notion of sending more data than is initially present is handled by increasing the size of the coding vector. There are likely more implementation details involved in implementing this feature. In this way the model seems to be more applicable to a p2p file downloading network, where the initial data is not updated over time.

The paper also does not cover standard gossip approaches in which nodes become disinterested in spreading information as they become unsuccessful at it. Even with these simplifying assumptions, including having a complete network for communication, the analysis is quite difficult. This suggests the idea that more powerful tools for analysis may have to be developed before similar results can be applied on more interesting network topologies. This path of research is suggested at the end of [6]. Fortunately, the paper provides a clear and rigorous basis on which to further develop this topic.

**3.4. Distributed Storage.** We can see that the random coding approach is also a kind of erasure code. These codes have been traditionally applied in distributed storage systems. In [17] the implications of using random network coding for distributed storage are studied.

It is found that the amount of central control can be significantly reduced in such a system, while retaining the benefits of an erasure code system.

We set up a simple system and compare network coded storage to uncoded storage and erasure code based storage. The model here is as follows:

- The file is broken up into  $k$  pieces (This is denoted by  $m$  in the paper)
- We distribute  $p$  packets to each node(storage element) (Denoted by  $k$  in the paper)
- $r$  counts the number of storage elements
- A user, who is separate from the nodes, wishes to obtain the file from the nodes

Some pseudocode for this setup would be as follows for the network coded version:

---

**Algorithm 3** NetworkCodingStoreServer

---

```

for  $s \in StorageElements$  do
  for  $i \in [1, \dots, p]$  do
    repeat
       $v \leftarrow random(F_q^k)$ 
       $b \leftarrow \sum_{i=1}^k v_i \cdot m_i$ 
    until store  $(v, b)$  at  $s$ 
  end for
end for

```

---

The repeat until loop retries until a successful store happens. A store may be unsuccessful if the vector generated is linearly dependent with the vectors the storage element already has. Likewise in the uncoded approach following, the store will be unsuccessful if the storage element already has the piece that is being stored. Here we show the pseudocode for the uncoded approach.

---

**Algorithm 4** UncodedStoreServer

---

```

for  $s \in StorageElements$  do
  for  $i \in [1, \dots, p]$  do
    repeat
       $g \leftarrow random(M)$ 
    until store  $g$  at  $s$ 
  end for
end for

```

---

Then the erasure coded approach:

**Algorithm 5** ErasureCodedStoreServer

---

```

for  $s \in \text{StorageElements}$  do
  for  $i \in [1, \dots, p]$  do
     $g \leftarrow \text{getRandomCodeWord}(M)$ 
    store  $g$  at  $s$ 
  end for
end for

```

---

An erasure code produces  $k(1 + \beta)$  blocks such that any  $k\gamma$  unique blocks in this set can be used to produce  $M$ . `getRandomCodeWord(M)` uniformly and independently chooses one of the blocks produced by the code.

Then we have the client code, which simply does the opposite, getting all the pieces from each element, then deciding if it can decode or not:

**Algorithm 6** Client

---

```

for  $s \in \text{StorageElements}$  do
   $data \leftarrow data \cup \text{packets}(s)$ 
end for
if can produce  $M$  from data then
  success
else
  fail
end if

```

---

The exact details on what the pieces are and how the client can decide if  $M$  can be produced from the data are specific to the approach used.

We then present the bounds from [17]:

**Proposition 3.11.** [17] *Since  $D_r \succ_{st} \min\{k, \text{Binomial}(pr, 1 - \frac{1}{q})\}$ , if we let  $r_c = \min\{r : D_r = k\}$  then*

$$pE[r_c] \geq \frac{kq}{q-1}$$

Here we get the stochastic bound by using 3.1. Each vector is a trial, and we have  $kr$  of these, with the lemma giving the probability of success. Since we can succeed at most  $m$  times, we add the min function to correct for it. Then  $r_c$  is the first round in which the receiver is able to decode, and we can get the expected value by applying expected values to the bounds.

**Proposition 3.12.** [17] *Let  $Z(r) = k - D_r$  be the residual degrees of freedom after connecting to  $r$  storage elements.*

$$\text{If } \frac{pr}{k} \geq 1 + \frac{2 \log_q(k)}{k} \text{ then } Pr(Z(r) = 0) \geq 1 - \frac{1}{k}$$

This follows from a result about a random matrix over a finite field. The set of  $kr$  vectors is considered as a single matrix, and the result describes the probability that the matrix will have a certain rank.

**3.4.1. Analysis of uncoded and erasure code approaches.** We start by looking at the erasure coded approach, in which we want  $k\gamma$  unique blocks out of  $k(1 + \beta)$ . We can see that the uncoded approach is a special case where  $\gamma = 1, \beta = 0$ .

**Proposition 3.13.** [17] *Let  $Z_m(r)$  be the number of additional pieces required to be downloaded after connecting to  $r$  storage elements, and let  $r_c$  be the random variable such that  $r_c = \min\{r : X_k(r) \geq k\gamma\}$  then:*

- (1)  $E[X] \sim k(1 + \beta)[1 - (1 - \frac{1}{k(1+\beta)})^{pr}]$
- (2)  $pE[r_c] \leq g(k) \sim k(1 + \beta) \ln\left(\frac{1+\beta}{1+\beta-\gamma}\right)$

Proving this result follows by using an inclusion-exclusion argument on the sets of blocks. We can see that we cannot use this result directly with the uncoded case, since we get an undefined result, however, we can use a sufficiently small  $\beta$  in order to get an estimate of the performance of the uncoded case.

3.4.2. *Remarks.* Again we build a table of the results:

	Linear Codes	Erasure Codes
$E[X_m] \geq$	$E[\min\{k, \text{Binom}(pr, 1 - \frac{1}{q})\}]$	$k(1 + \beta)[1 - (1 - \frac{1}{k(1+\beta)})^{pr}]$
$pE[r_c] \leq$	$\frac{kq}{q-1}$	$k(1 + \beta) \ln\left(\frac{1+\beta}{1+\beta-\gamma}\right)$

Here we have parameters on both sides that can be manipulated in order to change the expected minimum number of storage elements necessary. We can modify  $\beta$  to get the same gains as modifying  $q$ . However, increasing  $q$  requires no additional space, whereas modifying  $\beta$  can require more storage at the server depending on the erasure code used, such as Reed-Solomon codes.

The analysis on distributed storage here is in a peer-to-peer setting. The results concentrated mostly on the probability of a downloader being able to reconstruct the original file. This is a situation in which it is acceptable if the file cannot be reconstructed with low probability. This would not be acceptable in a long term storage solution. An issue that might be important in a long term storage situation would be the ability of the server to distribute packets in a way that retains a high probability of reconstructing the file despite storage server crashes. In a long term storage solution, the central server could easily be used to ensure that it would at least always be possible to reclaim the file if no storage servers crash by doing a linear independence check.

**3.5. Security for randomized network coding on P2P networks.** Avalanche[8, 9], is very similar to a gossip based information distribution network, however, the assumption that each node has one message that it wishes to spread is not as valid. In this case, we have some collection of original servers that have the whole file.

In this case a new problem shows up when we cannot trust the peers in our network. There may be some that try to inject incorrect data, which will result in a corrupted file when it is decoded.

The solution applied in this situation by Bittorrent [5] and other P2P schemes, is simply to use a cryptographic hash function on each packet. Since it is infeasible to produce collisions for these hashes, we can publicly distribute them as long as the peers are sure that the hash is genuine. In particular, Bittorrent applies a SHA-1 hash function.

Since these hashes are also computed quite quickly, they are a practical and efficient solution to the problem of forged packets in a P2P network. However, in general, these hash functions are not compatible with the linear coding done in network coding environments. What is needed is a hash function that is compatible with linear coding, known as a homomorphic hash function.

3.5.1. *Secure random checksum: a homomorphic hash function.* Unlike the previous applications, in a P2P environment, other nodes may not be trusted. Thus the creators of Avalanche present their solution, SRCs, or secure random checksums.

SRCs work as follows: we randomly generate a number seed, then:

$$SRC(seed, b_i) = b_i \cdot srcv(seed) = (b_{i,1}, \dots, b_{i,bsz}) \begin{pmatrix} r_1 \\ \vdots \\ r_{bsize} \end{pmatrix} = \sum_{k=1}^{bsize} r_k \cdot b_{i,k}$$

Where  $srcv(seed)$  is a vector of random numbers generated from seed and  $b_i$  is a particular block of data as defined earlier. We then compute  $SRC(seed, B) = B \cdot srcv(seed)$  which is the same as the vector  $(SRC(seed, b_1), SRC(seed, b_2), \dots)$ , and securely send  $(seed, SRC(seed, B))$  to the peer. Now we can use the homomorphic property of this hash to verify any packet that is received.

**Proposition 3.14.** *If  $cb$  is authentic, then  $cv \cdot B = cb$  and*

$$cv \cdot B \cdot srcv(seed) = cv \cdot SRC(seed, B) = cb \cdot srcv(seed) = SRC(seed, cb)$$

Verification proceeds as follows:

- Receive  $(cv, cb)$  from a peer.
- if  $cv \cdot SRC(seed, B) = SRC(seed, cb)$  then it is potentially correct, otherwise the packet is rejected.
- The test is repeated for several seed values, usually around 10

The security of this scheme depends on the difficulty of generating a collision when the seed value is not known. This difficulty depends on the field size, since one brute force method can simply generate a block iterating through all the possible choices of the first symbol, while keeping all the other symbols 0. For a field size of  $2^8 = 256$  for example, it would be quite easy to produce a collision for a single SRC.

All the operations involved are multiplications and additions, so generating and checking SRCs can be done fairly quickly. However its simplicity also means that it is a fairly weak security scheme. If the seed is known to attackers, then a fake block can be easily generated. This means that SRCs must be secret and a new set must be regenerated each time for each node in the system. This requires the server to have the full set of data, which is sometimes a limiting factor.

3.5.2. *Stronger hash functions.* The SRC solution has a weakness in that collisions are quite likely to be randomly generated. However, the property of SRCs that they are homomorphic with respect to linear functions, is the main requirement. Another homomorphic hash function is presented in [10] which is now secure enough to be computed once and publicly distributed.

This function requires that coding be done in the modular fields of prime size rather than the standard Galois fields with power of 2 sizes. This allows us to compute a hash of the following form:

$$h(b_i) = \prod_{k=1}^m g_k^{b_k, i} \text{ mod } r$$

Where  $g$  is a vector of numbers with certain properties. The security of this function comes from the intractability of the discrete logarithm problem. The

collection of  $h(b_i)$  values can then be securely but publicly transmitted to all the nodes in the system. An encoded block  $e$  can then be checked at a node by seeing if the following condition holds:

$$h(e) = \prod_{k=1}^m g_k^{e_{k,i}} \bmod r = \prod_{i=1}^n (h(b_i))^{c_i} \bmod r$$

Since checking this hash requires exponentiation rather than just multiplication and addition, it is more expensive to compute than SRCs. So instead of checking each individual coded block, coded blocks are randomly combined, then this new coded block is checked. There is now an added possibility of bad blocks slipping through, but it is still quite small.

This approach is combined with a cooperative warning system in which nodes notify each other when a bad block is found. This warning system is protected against false warnings by SRCs. Unfortunately, by using SRCs, we have to again ensure that the trusted server has the original file and has secure connections to peers. If faster homomorphic hash functions are found, it may be possible to fully eliminate this need.

Here we end our survey of p2p applications of distributed randomized network coding, and move on to the problem it was originally created for: multicast.

#### 4. MULTICAST

Multicast in networks is becoming increasingly important for information distribution. Streaming media and large file transfers are only a few examples. Here we present some theoretical results about applying the distributed randomized network coding approach to the multicast problem.

**4.1. Models and notation for Multicast networks.** We base this presentation on [14]. The concept is similar to that for peer networks, but small differences require a little more detail in notation:

- The network is a directed graph  $G = (V, E)$ 
  - For every  $e = (i, j) \in E$ , we define  $origin(e) = i$  and  $dest(e) = j$
  - We assume that  $G$  is irreflexive, so for all  $e \in E$ ,  $origin(e) \neq dest(e)$
  - For every  $v \in V$  we define  $inputs(v) = \{e \in E | dest(e) = v\}$ ,  $outputs(v) = \{e \in E | origin(e) = v\}$
  - The convention will be that  $i, j, l, e$  range over edges, while  $v$  ranges over vertices
- We can extend this graph to a multigraph in order to model higher bandwidth links.
- Every output is a linear coding of the inputs received
- Rather than transmitting messages, we transmit processes, or sequences of binary data. This gives the presentation more of an information theory feel.
  - Source processes are denoted  $X_1, X_2, \dots, X_r$
  - Each edge  $e \in E$ ,  $Y(e)$  is the output process for that edge, which will be received by  $dest(e)$  from  $origin(e)$

**4.2. Analysis of Randomized Coding for Multicast networks.** The peer networks that we have seen above are recent applications of randomized network coding. However, the original motivation for network coding in general was to solve multicast problems. The multicast case is often more complex, as we need to consider dynamic network issues, rather than just the problem of collecting enough correct packets. We can apply the analysis techniques used above to the multicast problem, and that is in fact where these questions originate from.

In analyzing these situations, the framework provided by Koetter and Medard in [14] is very powerful. Since we restrict our attention to linear codes, we can represent the effect of network coding in the network in a matrix form. Then we can use standard results about random matrices and other results in linear algebra to obtain probabilistic bounds. The concept seems intuitive, but the notation can get a quite heavy.

We start with the delay-free model, which is a lot simpler.

**4.3. Delay-free multicast.** In this model, presented in [12, 14], we need to capture the effect of the source nodes, the intermediate nodes, and the receiver nodes. These are not necessarily disjoint. We first notice that source and intermediate nodes can be handled together, so for each edge  $j$ , we have a process  $Y(j)$  representing the information being sent along that edge:

$$Y(j) = \sum_{i=1}^r a_{i,j} X_i + \sum_{l \in \text{inputs}(\text{origin}(j))} f_{l,j} Y(l)$$

We now have a matrix  $A$  of source process coefficients, and a matrix  $F$  of intermediate process coefficients. These matrices are indexed by the edges of the graph, not the vertices.

Next we handle the receivers. We want to know that receivers have received all the processes. We know this by giving each receiver  $v$  an output process corresponding to the source process  $X_i$ , which we will denote  $Z(v, i)$ . The receivers produce these output processes in the same manner as their outputs, by linearly combining them as follows:

$$Z(v, i) = \sum_{l \in \text{inputs}(v)} b_{v,i,l} Y(l)$$

It is helpful to remember that these receiver output processes are not transmitted over links, they can be interpreted as the data that the user or program at the node sees.

We will know if the receivers have received the source processes if we can make the processes  $Z(v, i) = X_i$  for all  $i$ . This gives us a collection of matrices  $B_v$ . We form the matrix  $B$  by stacking these matrices in block form.

$$B = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_d \end{bmatrix}$$

Finally, we have all the coefficients needed to represent the network coding done in this case.

**Definition 4.1.** A linear network code is a triple  $(A,F,B)$  as constructed above.

We then need a notion of a transfer matrix for a receiver node  $v$ . This will give us the receiver output processes of  $v$  in terms of the original input processes.

**Fact 4.2.** [14] *The matrix  $A(1 - F)^{-1}B_v^T$  is the transfer matrix for  $v$ .*

In the delay-free case, it is not really possible to reason about cycles, so we restrict our attention to acyclic graphs in presenting a main result of [11]. The proofs and results are taken from [12]. We want to look at the probability that a coding scheme in which all the coefficients are chosen randomly is successful.

In order to do that, we need to consider the transfer matrix for each of the receivers, in which case, the network code is successful if the transfer matrix of each receiver has full rank. This is clear because it means that each receiver can produce the source processes. We can determine this by looking at the determinant of the matrix. Since we are considering all of the coefficients in the system as variables to be determined later, the determinant becomes a multivariable polynomial.

The transfer matrix polynomial is difficult to handle, so we use the following result:

**Theorem 4.3.** *For a network code  $(A,F,B)$ , the transfer matrix  $A(1 - F)^{-1}B_v^T$  is nonsingular (has full rank) if and only if the matrix  $\begin{bmatrix} A & 0 \\ I - F & B_v^T \end{bmatrix}$  is nonsingular.*

The proof follows by applying a series of properties of determinants. Now we need to find out what kind of polynomial  $P$  is in order to bound the probability that it is 0.

**Lemma 4.4.** *For a network code  $(A,F,B)$ , the determinant  $\begin{vmatrix} A & 0 \\ I - F & B_v^T \end{vmatrix}$  is a multivariable polynomial  $P$  in variables  $a_{i,j}, f_{l,j}, b_{v_i,l}$ , each of which has a maximum degree less than or equal to  $v$ . with a total degree less than or equal to  $d\eta$*

Then we bound the probability that polynomials of this type are 0.

**Lemma 4.5.** *Let  $P$  be a multivariable polynomial of degree less than or equal to  $d\eta$ , in which the largest exponent of any variable is at most  $d$ . If the variables are chosen independently and uniformly at random from the same field, then  $\Pr(P(X) \neq 0) \geq (1 - \frac{d}{q})^\eta$  for  $d < q$ .*

The proof of this result is fairly involved, but intuitively we can factor the polynomial  $\eta$  different ways, and each of these ways has a factor that has a probability  $d/q$  of being 0. If all of these factors are non-zero then the whole polynomial is non-zero.

**Theorem 4.6.** *For a feasible multicast connection problem on an acyclic delay-free network, if some code coefficients are chosen independently and uniformly, with fixed coefficients preserving feasibility, the probability that all receivers can decode all the source processes is at least  $(1 - d/q)^\eta$  for  $q > d$  where  $d$  is the number of receivers and  $\eta$  is the maximum number of links receiving signals with independent randomized coefficients.*

*Proof.* This follows from the previous lemmas. □

We can see that when the ratio  $d/q$  is small, then the probability of success is quite high. This may be offset by a large  $\eta$ , but usually this will not be a problem. Now we have to deal with the general case of delays and cycles.

**4.4. Multicast with Delay.** The case where delay is introduced is somewhat more complicated, because differences in path lengths may mean that information has to be stored and that we must introduce a notion of time into the coding scheme. However, in this model, we can allow cycles in the network. First, we introduce a time variable  $t$ , and modify the equations from the acyclic case. We use the presentation of this setup from [13].

The first equation involving the processes output on each link is simple enough to modify, since we know from [14] that memoryless operation is sufficient for multicast at intermediate nodes.

$$Y_j(t+1) = \sum_i^r a_{i,j} X_i(t) + \sum_{l \in \text{inputs}(\text{origin}(j))} f_{l,j} Y_l(t)$$

Modifying the receiver output is a little more complicated, however. We give each receiver process a memory of  $\mu$  previous inputs, and also  $\mu$  previous outputs. We can then form our next receiver output as a combination of all of this previous data, and our current data (where  $u = 0$ ).

$$Z_{v,i}(t+1) = \sum_{u=0}^{\mu} b'_{v,i}(u) Z_{v,i}(t-u) + \sum_{l \in \text{inputs}(v)} \sum_{u=0}^{\mu} b''_{v,i,l}(u) Y_l(t-u)$$

Note that we now have two different B matrices, and also, all the terms now depend on  $t$ . In order to reason algebraically about this new situation, we must change  $t$  to a delay variable  $D$ . This is a plain scalar value, rather than any special type of structure. By creating polynomials in this variable  $D$ , we are able to reason about the properties over all time values at once.

First we have:

$$\begin{aligned} X_i(D) &= \sum_{t=0}^{\infty} X_i(t) D^t \\ Y_j(D) &= \sum_t Y_t(j) D^t, \quad Y_j(0) = 0 \\ Z_{v,i}(D) &= \sum_{t=0}^{\infty} Z_{v,i}(t) D^t, \quad Z_{v,i}(0) = 0 \end{aligned}$$

This shows the way we use polynomials to store the successive values of the processes at each time. The values show up as successive coefficients in higher powers of the delay variable  $D$ .

Now to represent the effect of coding, the intermediate links are straightforward again:

$$Y_j(D) = \sum_i^r a_{i,j} X_i(D) + \sum_{l \in \text{inputs}(\text{origin}(j))} f_{l,j} Y_l(D)$$

For the receiver outputs we get a remarkable simplification, by combining the two  $B_v$  matrices as follows:

$$b_{v,i,l}(D) = \frac{\sum_{u=0}^{\mu} D^{u+1} b''_{v,i,l}(u)}{1 - \sum_{u=0}^{\mu} D^{u+1} b'_{v,i}(u)}$$

Then we can simplify the receiver output processes to:

$$Z_{v,i}(D) = \sum_{l \in \text{inputs}(v)} b_{v,i,l}(D) Y_l(D)$$

This change comes by observing that previous values of  $Z$  are also just functions of  $Y$ , so no memory is needed to store the previous values of  $Z$ . So once again, our network code is represented as  $(A,F,B)$ , where  $B$  is the matrix formed by combining all the  $B_v$  matrices as before.

Then we present the result of [12]:

**Theorem 4.7.** *For a feasible multicast connection problem on a network with unit delay links and*

*a network code in which some or all of the coefficients are chosen independently and uniformly over a finite field, the probability that all receivers can decode the source processes is at least  $(1 - \frac{d}{q})^{\eta}$  for  $q > d$  where  $d$  is the number of receivers and  $\eta$  is the number of links carrying random combinations of sources or signals*

The proof proceeds essentially the same as the acyclic case, except we now have to account for the new delay variable added. Despite the added complication in modelling the problem, the bound is still the same.

4.4.1. *Remarks.* The connection between this model of randomized network coding and the distributed random network coding approach was not really emphasized in the papers. Since new random vectors are being generated constantly, we have a new instance of this problem each time a new packet is sent. It would be interesting to see if there is a way to maintain a successful code over time in order to reduce decoding time, while still keeping the benefits of the decentralized approach.

This analysis provides a helpful lower bound on the success probability of a random network code. We can see, as has been pointed out in several of the papers already mentioned, that field sizes of 16 and 32 bits will give fairly good success probabilities, since these field sizes are usually much larger than the number of receivers. Also, since the code is not fixed, but changes over time, it is acceptable to only have a high probability of success, as failures of the code will simply slow the network down.

## 5. CONCLUSION

Randomized network coding is a relatively new invention, originally created to solve the multicast problem. However, new applications in peer networks have surfaced. Though randomized network coding has simplified the implementation of random coding, it presents new challenges in analyzing its performance. For peer networks, changing the model requires different types of analysis. The gossip model was mainly concerned with dissemination time, while the storage model was concerned with the probability of successfully decoding the data. Security measures for such networks were also presented, though they were mainly for the

peer to peer file downloading network case. The models for analysis presented had many simplifying assumptions, yet the analysis was still difficult.

We then returned to the multicast case and gave an overview of how these situations are analyzed. This situation turns out to be more difficult, but mainly because in this situation it is possible to analyze arbitrary topologies. The major result is actually the formulation and modelling of the multicast problem in terms of algebraic objects. From this formulation, the results follow by applying properties of these objects.

Throughout the papers presented, we have used algebraic formulations of the problems presented to analyze their characteristics. In most cases, the algorithms and the ideas behind them are far simpler than the techniques used to analyze them. The frameworks for analyzing p2p applications in [2] and that for analyzing multicast problems in [14, 13] have proven to be almost essential in making theoretical analysis of these situations possible in a clear and rigorous manner.

Even so, there are many more possibilities for analysis, especially for the p2p applications discussed. Hopefully continuing research in this area will shed new light on the performance of these applications and further uses of the distributed randomized coding method.

#### REFERENCES

- [1] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.
- [2] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE/ACM Trans. Netw.*, 14(SI):2508–2530, 2006.
- [3] Shiwen Chen, Oktay Günlük, and Bülent Yener. The multicast packing problem. *IEEE/ACM Trans. Netw.*, 8(3):311–318, 2000.
- [4] P. Chou, Y. Wu, and K. Jain. Practical network coding, 2003.
- [5] Bram Cohen. Bittorrent protocol specification. Feb 2007.
- [6] Supratim Deb, Muriel Médard, and Clifford Choute. Algebraic gossip: a network coding approach to optimal multiple rumor mongering. *IEEE/ACM Trans. Netw.*, 14(SI):2486–2507, 2006.
- [7] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM Press.
- [8] Christos Gkantsidis, John Miller, and Pablo Rodriguez. Comprehensive view of a live network coding p2p system. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 177–188, New York, NY, USA, 2006. ACM Press.
- [9] Christos Gkantsidis and Pablo Rodriguez. Network coding for large scale content distribution. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, pages 2235–2245 vol. 4, 2005.
- [10] Christos Gkantsidis and Pablo Rodriguez Rodriguez. Cooperative security for network coding file distribution. In *INFOCOM*. IEEE, 2006.
- [11] T. Ho, R. Koetter, M. Médard, D. Karger, and M. Effros. The benefits of coding over routing in a randomized setting, 2003.
- [12] T. Ho, M. Médard, J. Shi, M. Effros, and D. Karger. On randomized network coding, 2003.
- [13] Tracey Ho, Muriel Médard, Ralf Koetter, David R. Karger, Michelle Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006.
- [14] Ralf Koetter and Muriel Médard. An algebraic approach to network coding. *IEEE/ACM Trans. Netw.*, 11(5):782–795, 2003.
- [15] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [16] S. M. Ross. *Stochastic Processes*. John Wiley and Sons, 1983.

- [17] M Medard R Koetter S Acedanski, S Deb. How good is random linear coding based distributed networked storage?, 2003.
- [18] Peter Sanders, Sebastian Egner, and Ludo M. G. M. Tolhuizen. Polynomial time algorithms for network information flow. In *SPAA*, pages 286–294. ACM, 2003.