How Good is Random Linear Coding Based Distributed Networked Storage?

Szymon Acedański, Supratim Deb, Muriel Médard, Ralf Koetter

***Abstract*— We consider the problem of storing a large file or multiple large files in a distributed manner over a network. In the framework we consider, there are multiple storage locations, each of which only have very limited storage space for each file. Each storage location chooses a part (or a coded version of the parts) of the file without the knowledge of what is stored in the other locations. We want a file-downloader to connect to as few storage locations as possible and retrieve the entire file. We compare the performance of three strategies: uncoded storage, traditional erasure coding based storage, random linear coding based storage motivated by network coding.**

We demonstrate that, in principle, a traditional *erasure coding* based storage (eg: Reed-Solomon Codes) strategy can almost do as well as one can ask for with appropriate choice of parameters. However, the cost is a large amount of additional storage space required at the centralized server before distribution among multiple locations. The *random linear coding* based strategy performs as well without suffering from any such disadvantage. Further, with a probability close to one, the minimum number of storage location a downloader needs to connect to (for reconstructing the entire file), can be very close to the case where there is complete coordination between the storage locations and the downloader. We also argue that an uncoded strategy performs poorly.

I. Introduction

In this paper, we concern ourselves with a key question that may arise in designing efficient distributed file systems. The question we ask is: how to store files in a large distributed system in an efficient manner? The problem has the constraint that, even though the total memory of all the nodes combined may be sufficient, the memory available at any particular node is limited. Suitably designed file-distribution strategies can find application in content-distribution networks, peer-to-peer networks, and also distributed libraries.

The most common method by which files are transferred on the Internet is the client-server model. A central server sends the entire file to each client that requests it. The clients only speak to the server, and not to each other. The main advantages of this method are that it's simple to set up, and the files are usually always available since the servers tend to be dedicated to the task of serving, and therefore are always on and connected to the Internet. The client-server model has a significant problem with files that are large or very popular. Namely, it takes a great deal of bandwidth and server resources to distribute such a file, since the server must transmit the entire file to each client. This may result in very slow download speed.

The concept of mirrors partially addresses this shortcoming by distributing the load across multiple servers. However, a great deal of coordination and effort are required to set up an efficient network of mirrors.

Another method of transferring files utilizes a peer-to-peer network. Systems such as Kazaa[9], Gnutella[5], e-Donkey, Direct Connect, etc. are examples of peer-to-peer networks. In most of these networks, Internet users trade files by directly connecting one-to-one. The advantage of this method is that files can be shared without having access to a server. Bit-torrent [12] is a protocol designed for transferring files in such manner. The users connect to each other directly to send and receive portions of the file. There is a central server (called a tracker) which coordinates the action of all such peers. The tracker manages connections, but does not have any knowledge of the contents of the files being distributed, and therefore a large number of users can be supported with relatively limited tracker bandwidth. In bit-torrent, selecting the pieces to download in a good order is very important for good performance. Several *adhoc* strategies are described in [12]. In general, the idea is to ensure that, the different pieces should be more or equally spread in the system as the system evolves.

The model (which we describe later) we have in mind is depicted in Figure I. While the dynamic behavior of a system can vary depending on the application one has in mind, our goal in this paper to study efficient strategies for distributing a large file (or multiple large files) into various storage locations. By efficient strategies, we mean, strategies that have the following properties:

- *Stateless*: The centralized server does not have to maintain any states of the storage locations, in terms of what they are storing. Such a property is highly desirable in environments where the number

S. Acedański is with the Warsaw University, Poland; e-mail:accek@mimuw.edu.pl. S. Deb and M. Médard are with the Massachusetts Institute of Technology, Cambridge, MA, USA; e-mail:{supratim,medard}@mit.edu. R. Koetter is with the University of Illinois at Urbana-Champaign, IL, USA; e-mail:koetter@uiuc.edu

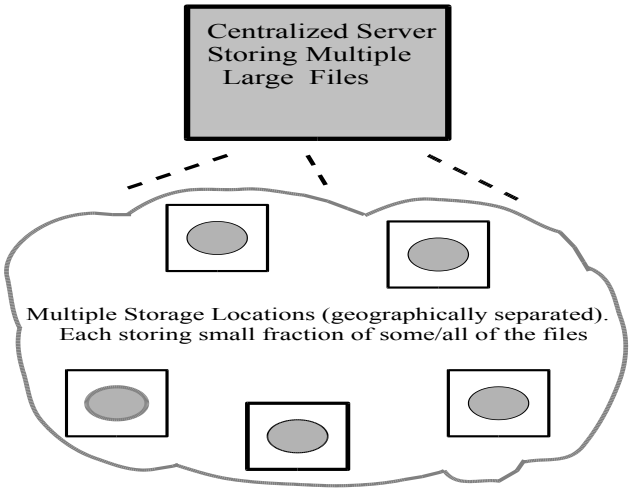

Fig. 1. Network with Distributed Storage Elements

of distinct storage location is quite large. The file-distribution protocol of bit-torrent is not *stateless*.

- *Independence*: What a storage location stores is independent of what might be stored in any other location. The key point is, there is absolutely no coordination between the storage locations, either direct or indirect (through the centralized server). This property clearly goes hand in hand with the previous one.
- *Performance guarantee*: While the above properties are highly desirable, if possible, the system should provide strong performance guarantee to a downloader. Since we are considering randomized strategies, probabilistic performance guarantees are more appropriate for the system we are studying. In other words, with a probability close to one, the downloader should complete the download after connecting to as few storage locations as possible. While the exact performance measure of a system may depend on the specific application (peer-to-peer, content distribution etc) in mind, we believe such a notion of performance should translate into the very good performance of the system.

The question we ask is: is it possible to attain all the above in a system? We demonstrate that, certain simple and easy to implement coding strategies can indeed satisfy all the properties with appropriate tradeoffs. The tradeoff is either in terms of complexity of decoding algorithms required at the downloader or additional storage space required at the centralized server.

Independent of our work, the authors in [4] have also proposed and compared strategies very similar to ours and have reported detailed simulation experiments for content distribution network. We differ from them in carrying out detailed mathematical analysis and simulations of a generic model (see Figure I) that might have wider applicability.

We primarily analyze and compare three strategies for storing: an *uncoded random storage*, traditional *erasure coding based storage*, and *random linear coding based storage*. The *random linear coding* based storage we discuss in this paper is motivated by the utility of *random linear codes* in the emerging area of network coding. Network coding can be viewed as a vast generalization routing where packets (which are nothing but a collection of bits) are treated as algebraic elements, rather than mere transportation or storage elements. Thus, packets, just like any other algebraic entity, can be operated upon. It was shown in [11] that, linear network coding can achieve the min-cut bound in networks with multicast flows. There is a significant recent work on network coding [8], especially on the algorithmic aspects of construction of linear network codes [1], [10]. In [6], [7], the authors proposed the novel idea of *random network coding*. A common theme running across most of this work is to demonstrate that a *linear coding* based approach can improve the capacity of various networks. In this paper, we will argue that *random linear coding* based schemes motivated by network coding can have wider applicability in uncoordinated distributed storage.

II. Model, Performance Measures, and Storage Strategies

A. Model and performance measures

We consider a large file which is broken into m pieces. Suppose there are multiple nodes or distributed memory elements each of which can store k of the m pieces (see Figure I). We refer to the distributed memory elements as *peers* henceforth. *We assume that no peer has any knowledge about what the other peers have stored.* This is a key assumption in our model. An implication of this assumption is that, even if there is a central controller to distribute the pieces, the controller does not differentiate between the peers, and does not need to maintain a state of the contents of all the peers. Thus the peers do not coordinate for storing pieces of the file. In fact, we will demonstrate that a coding based storage makes redundant any need for coordination between the peers for storing different pieces. A downloader completes its download once it gathers each of the m elements or has enough *information* to recover the m pieces that constitute the entire file. The downloader can connect to a fixed number of peers at a time.

Before we describe the three strategies we describe the performance measures we are interested in. Suppose, the downloader got opportunities to connect to r of the peers after spending some time in the system (this can be over single or multiple connection instants depending

on the value of r and the maximum number of peers the downloader can connect to). The questions we ask are the following:

- For any given fraction x, what is the probability that r peers can provide sufficient information for the downloader to complete at least x fraction of the download? This is clearly a function of m, k, and r.
- What is the mean fraction of overall download that can be availed from r different peers for different values of r?

We will also address a few other closely related performance measures in our results.

B. Strategies for Distributed Networked Storage

We consider the following strategies for storing parts of the file in the various locations:

Uncoded Random Storage: This is a simple scheme where each peer stores k different pieces out of the m pieces at random. Thus there are $\binom{m}{k}$ ways of storing data elements in a peer. If the peers choose their contents without the knowledge of what other peers have, this strategy is a natural one if coding is not allowed.

Traditional Erasure Coding Based Random Storage: In this scheme, the m distinct blocks are encoded at the source using appropriate erasure codes. The m original blocks are encoded to generate $m(1+\beta)$ encoded blocks so that any $m\gamma$ out of the $m(1+\beta)$ blocks are sufficient to decode the entire file. Here, $1/(1+\beta)$ is the code-rate and the relation between γ and β depends on the specific code. Now, each storage location stores k out of the $m(1+\beta)$ encoded blocks at random. Two popular examples of erasure-codes are Reed-Solomon codes and Tornado codes. The entire file can be decoded once a downloader has access to any different $m\gamma$ number of chunks.

Random Linear Coding based Storage: In this scheme, we view the m pieces of the file as elements in $\mathbb{F}_q^s$, i.e., vectors of size s in a field of size q. Thus, if we denote the chunks by $c_i,\ i = 1, 2, \ldots m$, then each peer stores k random linear combinations of c_i's. More specifically, if the elements with a particular peer are $f_1, f_2, \ldots f_k$, then a typical element f_i can be represented as

$$f_i = \sum_{j=1}^{m} \beta_j\ c_j, \quad \Pr(\beta_j = \beta) = \frac{1}{q}\ \forall \beta \in \mathbb{F}_q\ .$$

Further each peer also stores the associated vector $(\beta_1, \beta_2, \ldots, \beta_m)$ for each of the k pieces. We also call this the associated *code vector*. This will take an additional storage space of $km\log_2(q)$ bits. This is typically a small number compared to each piece of the broken file. For example, bit-torrent breaks up a file into pieces of size 256 KB. Suppose we have a file of size 25 MB and also let us suppose $q = 16$ (typically a smaller value may yield good results as we will see later). The file can thus be broken up into around 100 pieces, each of size 256 KB. Each randomly mixed piece in this scheme will require an overhead (to store the β_i's) of $m\log_2(q) = 400$ bits or 50 bytes/piece. Thus the additional storage space required in percentage is $100 \times 50\ bytes/256\ KB \approx 0.02\%$. This is indeed a negligible overhead, especially when we consider the benefits discussed in the result section. For reconstructing the entire file, it is sufficient that the dimension of the code-vectors with the downloader is m.

We now proceed to analyze the different strategies. We skip many of the proofs for want of space. In the following, we use the notation "$f(m) \sim g(m)$" to mean $\lim_{m\to\infty} f(m)/g(m) = 1$ where f and g are functions of m, the number of pieces the file is broken into.

III. Uncoded Random Storage

A. Exact Analysis of the Scheme

We first show some simple calculation for the *Uncoded Random Storage* mechanism. Fix r, the number of peers. Let S be a given set of pieces. We also denote by M the set of all the pieces. We first note that,

$$\begin{aligned} &g(S; |S| = j) \\ &\triangleq \Pr(\text{no element of the set } S \text{ features in the} \\ &\qquad r \text{ peers} \mid |S| = j) \\ &= (\Pr(\text{a particular peer has pieces only from } M \setminus S))^r \\ &= \left(\frac{\binom{m-j}{k}}{\binom{m}{k}}\right)^r\ . \end{aligned}$$

Note that, $g(S) = 0$ for $|S| > m - k$ since there are at least k distinct pieces with the peers. Let Y be the random variable denoting the exact number of pieces of the file missing in the r peers. Using an inclusion-exclusion argument [3], it follows that

$$\begin{aligned} \Pr(Y = y) &= \sum_{j=y}^{m-k} (-1)^{j-y} \binom{j}{y} \sum_{S:|S|=j} g(S) \\ &= \sum_{j=y}^{m-k} (-1)^{j-y} \binom{j}{y} \binom{m}{j} \left(\frac{\binom{m-j}{k}}{\binom{m}{k}}\right)^r \end{aligned} \quad (1)$$

The preceding completely characterizes the distribution of Y. Denoting by X the random variable the fraction of overall file available from the r peers, we can calculate all relevant statistics about X by noting that $\Pr(X \geq x) = \Pr(Y \leq m(1-x))$.

B. Asymptotic Analysis and Approximate Performance Measures

We first derive a key proposition which sheds more light into the performance of the scheme.

Proposition 3.1: Let $Z_m(r)$ be the number of additional pieces required to be downloaded after connecting to r peers, and let $X_m(r)$ be the number of available chunks after connecting to r peers. We, then have,

$$\mathbb{E}[X_m(r)] \sim m[1-(1-\tfrac{1}{m})^{kr}]$$

Let $r(s)$ be the minimum r such that $\mathbb{E}[Z_m(r)] = s$. Then, we have

$$kr(s) \sim m\ln(\tfrac{m}{s})$$

The proof of this is a special case of the proof of Proposition 4.1.

We now present approximation for the different performance measures based on the insight gained from Proposition 3.1.

Let $r(x)$ and $f(r)$ denote the following.

$r(x) =$ Number of peers to be connected so that the average fraction of download completed is x

$f(r) =$ The mean fraction of download completed after connecting to r peers.

Under the approximation that the mean number of missing chunks is roughly $m(1-1/m)^{kr}$, we obtain the following approximation for $r(x)$.

$$r(x) \approx m\ln(\tfrac{1}{1-x})$$

We also have the following approximation for $f(r)$.

$$f(r) = 1-(1-\tfrac{1}{m})^{kr}$$

Furthermore, the mean number of peers that need to be connected so that the entire download is completed can be easily seen to be at least $m\ln(m)/k$.

IV. Traditional Erasure-coding Based Random Storage

A. Exact Analysis of the Scheme

The starting point of the analysis of this is quite similar to the one for uncoded random storage except for a few differences. However, we shall soon see that the asymptotic behavior (when m is large) is very different which clearly brings out the advantages compared to an *uncoded random storage*.

Recall, that erasure coding on the m blocks would convert the chunks of the file into an encoded set of $m(1+\beta)$ blocks so that any $m\gamma$ out of the $m(1+\beta)$ blocks are sufficient to decode the entire file.

In the following, fix r the number of peers a downloader has connected to. Let Z be the number of chunks that remain to be downloaded.

Let Z denote the number of additional chunks a downloader needs for completing the download after having connected to r peers. We can now readily give an expression for the distribution of of Z from (1) substituting $m(1+\beta)$ for m and $m(1+\beta-\gamma)+Z$ for Y. In the following, we let $\theta = 1+\beta-\gamma$.

$$\Pr(Z=z) = \begin{cases} \sum_{j=m\theta+z}^{m(1+\beta)-k}(-1)^{j-m\theta-z}\binom{j}{m\theta+z}\binom{m(1+\beta)}{j}\left(\frac{\binom{m(1+\beta)-j}{k}}{\binom{m(1+\beta)}{k}}\right)^r \\ \quad \text{if } z>0, \\ \sum_{y=0}^{m\theta}\sum_{j=y}^{m(1+\beta)-k}(-1)^{j-y}\binom{j}{y}\binom{m(1+\beta)}{j}\left(\frac{\binom{m(1+\beta)-j}{k}}{\binom{m(1+\beta)}{k}}\right)^r \\ \quad \text{if } z=0. \end{cases} \tag{2}$$

The preceding completely characterizes the distribution of Z. Denoting by X the random variable the fraction of overall file available from the r peers, we can calculate all relevant statistics about X using $\Pr(X \geq x) = \Pr(Z \leq m\gamma(1-x))$.

B. Asymptotic Performance Measures

In this section we analyze the performance of a traditional erasure coding based storage policy in more detail. Our analysis will also shed light on the right choice of the coding rate $1/(1+\beta)$.

We start by presenting the following proposition.

Proposition 4.1: Let $Z_m(r)$ be the number of additional pieces required to be downloaded after connecting to r peers, and let $X_m(r)$ be the number of available chunks after connecting to r peers. We, then have,

$$\mathbb{E}[X_m(r)] \sim m(1+\beta)[1-(1-\tfrac{1}{m(1+\beta)})^{kr}]$$

Let r_c be the minimum r such that $\mathbb{E}[Z_m(r_c)] = 0$ (we hide the dependence of r_c on m). Then, we have

$$kr_c \sim m(1+\beta)\ln(\tfrac{1+\beta}{1+\beta-\gamma})$$

Remark 1: Suppose we have $\gamma = 1$ as with Reed-Solomon codes. Note that, with $\gamma = 1$, we have

$$kr_c \sim m(1+\tfrac{1}{2\beta}+o(\tfrac{1}{\beta}))\ ,$$

and thus, by choosing β large enough we can make kr_c/m arbitrarily close to one which is the best one can hope for. There are a couple of tradeoffs associated with this. First of all, a very large β comes at the cost of $(1+\beta)$ factor of additional storage space at the centralized server, where the file is stored before distributing it among the different locations. Secondly, the decoding complexity for decoding the encoded blocks goes up by a factor of $(1+\beta)$. □

We can infact provide a stronger probabilistic guarantee if the donloader connects to a few additional storage locations. It is captured in the following proposition.

Proposition 4.2: Let k (number of chunks each location stores) and r (number of locations a downloader has connected to) be jointly scaled with m so that $kr/m = \alpha$. Let Z_m be the random variable denoting the additional chunks required for the download to be completed.

Suppose $\alpha > (1+\beta)(1+\ln(\frac{1+\beta}{1+\beta-\gamma}))$. Then,

$$\lim_{m\to\infty} \Pr(Z_m = 0) = 1 \ .$$

In other words, the entire file can be decoded with probability approaching one as $m \to \infty$ if α is in the given range.

V. Random Linear Coding Based Storage

We next analyze the *random linear coding* based storage scheme. In this scheme, with r peers, there are a total of kr m-dimensional code-vectors available for the downloader. Recall that each of the code-vector represents a random mixture of the pieces. We can view the collection of these vectors as a $kr \times m$ matrix over $\mathbb{F}_q$. The complete file can be recovered once the dimension of this matrix is m. As the downloader gathers information from more and more peers, it gathers more and more independent code-vectors.

A. Exact Analysis

Let D be the random variable denoting the dimension of the subspace spanned by all the kr code-vectors with the r peers. Then, it can be shown that (see any standard reference on random matrix over finite fields),

$$\Pr(D = d) = \frac{\prod_{i=0}^{d-1}(q^{kr} - q^i)\prod_{i=0}^{d-1}(q^m - q^i)}{q^{mkr}\prod_{i=0}^{d-1}(q^d - q^i)} \qquad (3)$$

The preceding expression, though complicated, can be used to numerically derive all the performance measures. To gain more insight into the performance, we next show some very simple performance lower bounds.

B. Performance Lower Bound

We use the notation "$X \succ_{st} Y$" to imply $\Pr(X > a) \geq \Pr(Y > a)$ for all real a. We have the following.

Proposition 5.1: We have,

$$D \succ_{st} \min(m, \textit{Binomial}(kr, 1 - \tfrac{1}{q})) \ .$$

The proof is straightforward and follows immediately from Lemma 2.1 in [2].

The following corollary is immediate using standard applications of Chernoff bounds.

Corollary 5.1: Let $Z(r) = m - D$ be the residual degrees of freedom after connecting to r peers. If,

$$\frac{kr}{m} \geq \frac{1}{1-1/q}\left(1 + 2\sqrt{\frac{\ln(m)}{m}}\right),$$

then,

$$Z(r) = 0 \ \ \textit{with probability at least} \ \ 1 - \tfrac{1}{m} \ .$$

In other words, the download is complete almost certainly.

Remark 2: Let r_c be the minimum number of storage locations a downloader needs to connect to. Note that, by choosing a field size large enough, kr_c/m can be made arbitrarily close to one, which is the minimum one can hope for. In that sense, a large q has the same effect as the large β with traditional erasure coding. However, in this case we do not need any additional storage at the centralized location, where the file is stored before distribution. Further, in this case, we have a strong guarantee, i.e., kr_c/m is close to one with a very high probability. The tradeoff for a large field size is the minimally additional overhead with each chunk, and the complexity of the inversion of a matrix over a larger field size at the downloader. However, for all practical purposes the complexity is no more than than the inversion of a real matrix.

VI. Simulation Results

We next show some simulations results to illustrate the performance of various strategies. The purpose of the simulation is to provide a more detailed evaluation of the various strategies, and to demonstrate that the mean and lower bound based analysis are truly reflective of the performances of the various schemes. Simulation results for traditional erasure codes are with $\gamma = 1$, i.e., any m out of $m(1+\beta)$ encoded blocks are sufficient to decode the entire file. Reed-Solomon code is one such code.

The results are representative of simulations done with a wide range of parameters. In the results we show, we use the following parameters. We consider a single file broken into 50 chunks and each individual storage location can store 5 chunks, so that $m = 50$ and $k = 5$. This can be viewed as, say for example, any 1 GB file broken into chunks of size 20 MB and each storage location has a space worth 100 MB for each such file.

We show two kinds of plots in our results.

In our first set of plots in Figure 2, we show the probability that a given fraction of download is completed after a downloader has connected to $r = 10$ different storage locations. As we have 50 chunks and space of 5 chunks/location, the downloader needs to connect to a minimum of 10 locations in any event. We show the plots for uncoded random storage, erasure coding based strategy with $\beta = 4$ and $\beta = 8$, and random linear coding based strategy with field sizes $q = 3$ and $q = 7$. Note from the plots that, with a *random linear coding* based storage, more than 90% of the download is complete almost certainly even for a moderate field size of $q = 3$,

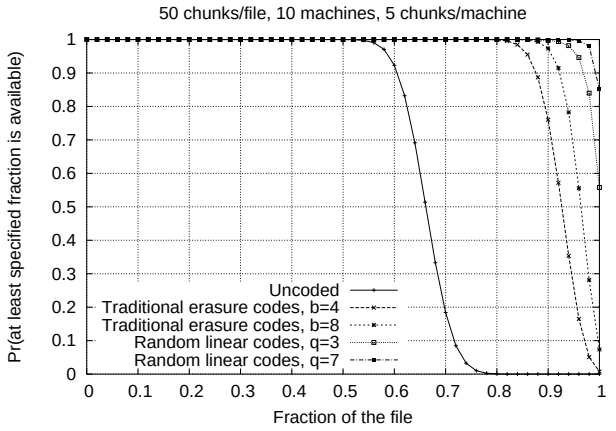

Fig. 2. Plots showing probability that a given fraction x of the file is available for downloaded for different values of x. The plots are shown for $r = 10$

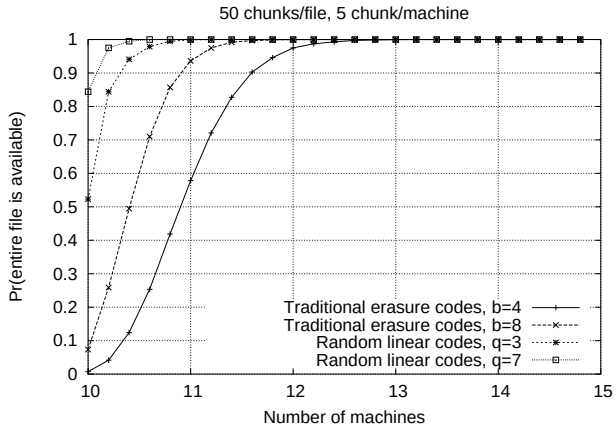

Fig. 3. Plots showing probability that the complete file is downloaded for different values of the number of peers contacted

and it is close to 95% for a field size of $q = 7$. The *erasure coding* based strategy also completes close to 90% of the download for $\beta = 8$. However, a field size of $q = 3$ requires an additional storage space around $10^{-4}\%$ at the storage locations for an original file size of 1 GB, whereas, an erasure coding based strategy with $\beta = 8$ requires an additional eight times, i.e., 9 GB storage space at the centralized file server. Also, note from the plots that an uncoded random storage performs very poorly and only completes around 55% of the download with a reasonable amount of certainty.

In Figure 3, we compare the probability that the entire download is complete for different values of r, i.e., the number of staorage locations a downloader connects to. The points in the plot for non-integer values of r correspond to collecting only some fraction of the chunks from a storage-location or peer. Note that, with *random linear coding* based storage, and for the values of the parameters considered, the entire download is complete almost certainly once a downloader connects to just 11 storage locations, which is one more than the minimum of 10 storage locations required in any event. For an *erasure coding* based strategy with $\beta = 8$, we almost see a comparable performance, i.e., 12 storage-locations are sufficient for decoding the entire file.

VII. Conclusions

We have demonstrated that, a *random linear coding* based storage, motivated by random network coding, makes the system behave as if the different storage-locations, which a downloader connects to, had complete coordination for storing the different chunks of the file. Thus the efficiency of the system is as close to the best one can ask for. Any traditional erasure coding based scheme can also perform almost as well, but requires a huge amount of additional storage space at the centralized file server. In this sense, a random linear coding based strategy can be called *erasure coding on the fly*.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46:1024–1016, 2000.

[2] S. Deb and M. Médard. Algebraic gossip: A network coding approach to optimal multiple rumor mongering. *M.I.T. LIDS Technical Report, Also submitted to IEEE Transactions on Information Theory*, April 2004.

[3] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. J. Wiley & Sons, New York, 1964.

[4] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. Technical Report MSR-TR-2004-80, Microsoft Research, 2004.

[5] Gnutella. http://gnutella.wego.com.

[6] T. Ho, M. Médard, M. Effros, and D. Karger. The benefits of coding over routing in a randomized setting. In *Proc. IEEE Symposium on Information Theory*, 2003.

[7] T. Ho, M. Médard, M. Effros, and D. Karger. On randomized network coding. In *Proc. 41st Allerton Annual Conference on Communication, Control and Computing*, October 2003.

[8] Network Coding Homepage. http://www.comm.csl.uiuc.edu/koetter/nwc/.

[9] KaZaA. http://www.kazaa.com.

[10] R. Koetter and M. Médard. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*, October 2003.

[11] S.-Y. R. Li, R. W. Yeung, and N. Cai. Linear network coding. *IEEE Transactions on Information Theory*, Februray 2003.

[12] Bit torrent file sharing protocol. http://bitconjurer.org/bittorrent/.