A Content Distribution System based on Sparse Linear Network Coding

Guanjun Ma* Yinlong Xu+ Minghong Lin* Ying Xuan*
Department of Computer Science and Technology, University of Science and Technology of China
Anhui Province-MOST Key Co-Laboratory High Performance Computing and Its Applications
+ Corresponding author's email: ylxu@ustc.edu.cn
* Authors' emails: {gjma,linmh,flames}@mail.ustc.edu.cn

***Abstract*— With network coding, intermediate nodes between source and destination node(s) encode the incoming packets into new ones and forward them to their outgoing links. The original content is decoded at the destination node(s). Recent theoretical results show that network coding is beneficial for peer-to-peer(P2P) content distribution. To evaluate the benefit of network coding, we implement a P2P content distribution system based on the *sparse linear network coding* method. In our system, we use the *Chord* protocol to construct the system topology. We determine the proper encoding density so as to reach a high probability of generating independent encoded blocks, and to reduce the computational complexity of encoding packets at each peer. To improve the system performance, we use the *encoding interval* to reduce the probability of transmitting linear dependent packets and *dependency test* to avoid accepting linear dependent packets possibly from cyclic topology. Lastly, we carry out extensive experiments to show in terms of average downloading time at peers, total distribution time and system throughput, the system with network coding slightly outperforms a BitTorrent-like non-coding system using the *local-rarest-first* chunk selection policy.**

I. Introduction

Network coding is originally proposed by R. Ahlswede *et al*. in 2000[1]. Under the network coding paradigm, intermediate nodes between the source and the receiver(s) encode incoming messages and forward the coded ones to their downstream nodes, which is different from the traditional service of relaying and replicating data messages only. T. Ho *et al.*[2] proposed random linear network coding, in which, the encoded packets are the linear combinations of the incoming packets and the coefficients are randomly selected from a finite field. Random linear network coding can be easily implemented in real networks and it greatly reduces the coding computational complexity. P. A. Chou *et al.*[3] further showed that random linear network coding can closely reach the optimal theoretical performance of network coding.

Content distribution is one of the most important applications of the Internet. It is implemented mainly based on Client/Server and/or P2P paradigm. P2P network has attracted a lot of attentions due to its scalability, decentralized control and the ease of management. A number of P2P content distribution systems have been developed, such as BitTorrent, eMule, KaZaA, and so on. For the past few years, BitTorrent has been highly regarded for its good performance and it is considered as the most popular P2P content distribution protocol. Under the BitTorrent content content distribution session, the source peer divides a file into some blocks and transmits these blocks to other peers. At the same time, peers also download blocks from other non-source peers while uploading their blocks. BitTorrent uses a combination of *random* and *local-rarest-first* schemes. With the *local-rarest-first* scheme, a peer requests a missing block and this block is selected among the rarest blocks of the peer's neighbors. This way, the file availability and performance can be greatly improved.

C. Gkantsidis *et al.*[4] indicated that the *local-rarest-first* scheme may ignore the global rarest blocks and may reduce the distributing speed in some cases. Based on random linear network coding, they built a BitTorrent-like content distribution system, Avalanche. Avalanche divides the file content, creates connections, downloads and uploads blocks in the same way as the BitTorrent protocol, but also encodes the incoming blocks before forwarding them to other peers. Authors in [4] claimed that the throughput of the Avalanche system is two to three times better than the non-coding system. However, the claim is based on simulation rather than implementation in real networks, in particular, one has to consider the high coding computational complexity. M. Wang *et al.*[5] implemented an experimental testbed based on sparse random linear coding. In their system, a fixed percentage of total blocks are selected to be encoded, and at least another fixed percentage of total blocks should be held by a non-source peer when it uploads blocks to other peers. The experimental results indicate that the system using network coding performs even worse than naive broadcast, which in turn has a lower performance as compare with the BitTorrent system. D.M. Chiu *et al.*[6] studied the performance of network coding for P2P content distribution in star networks. Their theoretical analysis shows that the throughput upper-bound of a system with network coding is the same as the system without using network coding.

To address whether there is any benefit in using networking coding, we implemented a P2P content distribution system with 50 nodes, based on a *sparse random linear coding* strategy, which is different from coding scheme used in [5]. In our system, we also use the *encoding interval* to reduce the probability of transmitting dependent blocks, as well as the *dependency test* to avoid receiving dependent blocks. Our experimental results show that network coding slightly outperforms non-coding. Our conclusion is neither so optimistic as the Avalanche[4], nor so pessimistic as [5].

The outline of our paper is as follows. In Section II and Section III, we describe the implementation of our system and

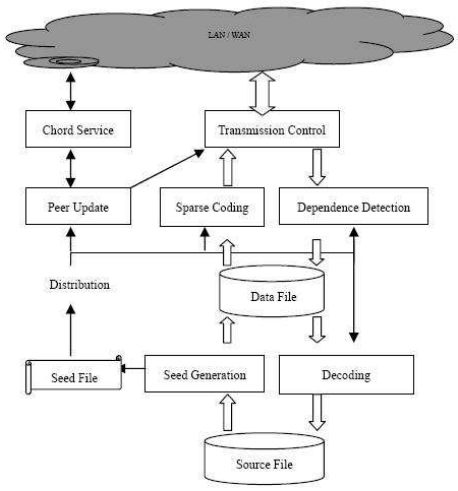

Fig. 1. The architecture of our system, including seven modules: Chord Service, Peer Update, Transmission Control, Seed Generation, Sparse Coding, Dependence Detection and Decoding

present the analysis and the experimental results. Section IV serves as the conclusion.

II. A Sparse Network Coding System

We implement a P2P content distribution system with 50 nodes. In our system, we use the *sparse linear coding* strategy based on C.Cooper's work[7] to reduce the encoding computational complexity and keep a high ratio of successful decoding. Fig. 1 illustrates the architecture of our system. In this section, we first outline the implementation of our system and discuss in detail some of the techniques used to improve the system performance. Experimental results will be discussed in later section.

A. Link Establishment

Our system consists of 50 nodes, and contains an IP list of all nodes joining the content distribution based on the *Chord*[8] protocol. The maximum links (or degree) of a peer is set to 10. If a peer has less than five links, it will randomly choose another peer in the IP list, and the peer will actively send a connection request and try to establish a link to that peer. Otherwise, it will not send connection requests and only passively wait for the connection requests from other peers. A link is successfully established as long as one peer has "*interest*" in the other peer (e.g., the other peer has more than 20 blocks of the file). The upper bound on the transmitting rate is set to 100 KB/s for each link to match the average transmitting rate in the Internet. A link will be cut in the case of both peers having received a fixed percentage of linearly dependent blocks from each other.

B. Sparse Coding, Dependence Detection and Decoding

To select a proper encoding density for low encoding computational complexity and high ratio of successful decoding, our system uses a sparse coding strategy based on [7] for random matrix over the Galois field. The following theorem can be deduced from [7] and is the theoretical basis of our sparse coding.

Theorem 1: Let $M = (m_{ij})$ be a random $n \times n$ matrix over the Galois field $GF(t)$ in which each matrix entry m_{ij} is identically and independently distributed, with

$$Pr(m_{ij} = r) = \begin{cases} 1-p & r = 0 \\ p/(t-1) & r \in [1, 2, ..., t-1], \end{cases} \quad (1)$$

then

$$\lim_{n\to\infty} Pr(M\ is\ non-singular)) = e^{-2e^{-d}} \prod_{j=1}^{\infty} 1 - t^{-j} \quad (2)$$

provided that $p \geq (\log n + d)/n$, where d is a nonnegative constant.

According to Theorem 1, one can assure that all encoded blocks generated by the same peer are linearly independent with high probability. If $p = (\log n + d)/n$, an $n \times n$ matrix will be non-singular with high probability, e.g. greater than 99.5% and 99.991% when $t = 2^8$, $d = 6$ and 10 respectively, for sufficiently large n. In this paper, we assume that the content is divided into n blocks and the encoding density is set to $p = (\log n + d)/n$ at the source peer. So the source peer can generate n independent blocks with high probability.

Let a non-source peer A hold k blocks $B_1, B_2, \ldots, B_k$ and generate l blocks $B_1^{'}, B_2^{'}, \ldots, B_l^{'} (l \leq k)$. Denote the $k \times n$ coefficient matrix of $B_1, B_2, \ldots, B_k$ as M, and the $l \times k$ local coefficient matrix of $B_1^{'}, B_2^{'}, \ldots, B_l^{'}$ at A as N, then the coefficients matrix of $B_1^{'}, B_2^{'}, \ldots, B_l^{'}$ will be $N \times M$. Because all peers accept only independent blocks in our system, therefore $rank(M) = k$. If $rank(N) = l$, $B_1^{'}, B_2^{'}, \ldots, B_l^{'}$ are linearly independent. According to Theorem 1, if $p = (\log k + d)/k$, the probability of $Rank(N) = l$ will be upper bounded by $e^{-2e^{-d}} \prod_{j=1}^{\infty} 1 - t^{-j}$ for sufficient large k and $B_1^{'}, B_2^{'}, \ldots, B_l^{'}$ will be linearly independent with high probability. Note that in practice, k may not be large enough to approach the precondition of Theorem 1. Therefore, we set $p = (\log n + d)/k$ at non-source peers in our system to keep a high probability of generating independent blocks. Our experiments show that $p = (\log n + d)/n$ at the source peer and $p = (\log n + d)/k$ at non-source peers are enough in practice. Too large a value of p will lead to a diminishing return in the independence probability, but at a cost of high encoding computational complexity.

Since the encoding processes among all peers are identical and independently distributed, it is difficult to ensure that encoded blocks from different peers are linearly dependent. Some factors that influence the linear dependency such as circles in the network topology will also bring linear dependency into distribution sessions. It is important to point out that linear dependency from these factors cannot be resolved by increasing the encoding density.

When a block arrives at a receiver peer, the coefficients of the block are extracted and added as a new row to a coefficient matrix in the the memory. Then, the receiver can decide whether to accept this block or to discard it after

checking the linear dependency of the coefficient matrix (i.e., via the Gaussian Elimination operation). In the evaluation of the system performance, we define *drop rate* at a peer as the fraction of dropped blocks to all received blocks. With the performance measure of drop rate, one can study the probability of transmitting dependent blocks with sparse coding. For those accepted blocks, one can add them as rows to a block matrix in the peer's secondary storage. This way, only the coefficients of a block, instead of the whole block, are stored in the memory for dependence checking, and the receiving process is greatly sped up. Also, the coefficient matrix stored in the memory can be converted to a triangular matrix (since it is possible that some diagonal elements become zero after elimination, we use column permutation to get a triangular matrix), which will speed up the elimination when a new block arrives. The original content can be successfully decoded at a peer when the peer has received n independent blocks. We implement most of our modules with Python except for the encoding and decoding modules. Since this script language is likely to lower the encoding and decoding efficiency, we resort to *C* and use their compiled *.dll* files as extended modules of Python so as to reduce the computational overheads of encoding and decoding operations.

C. Transmission Control

Transmission and reception of dependent blocks will greatly reduce the performance of content distribution via network coding. We introduce the following three techniques to reduce the probability of block dependency.

Neighbor selection: Once a link is established, its two end peers start inquiring each other about the number of blocks the other holds. If a peer holds too few blocks, it will transmit dependent blocks with high probability. Thus, except at the beginning or at the end of a content distribution session, no peer will request its neighbors containing less than $s = \log n + d$ blocks. However, considering that peers have few blocks at the beginning and need only a few new blocks to complete its receiving process at the end, these peers keep requesting all their neighbors for new blocks, regardless of how many blocks the neighbors hold. Although it is possible to produce more dependent blocks, our experimental results show that this strategy can significantly speed up the distribution process.

Circle elimination: Because of bidirectional links, circles may exist in a P2P system and are time-consuming to detect or discover. Circles in a network will incur a high dependent ratio and poor decoding performance. During the transmission, we adopt a block *dependency test* to reduce the impact of circles in a network topology. Let A and B be the two end peers of a link. If no less than 3% of blocks from A to B are dependent, B will stop requesting A for blocks for 30 seconds, and the *encoding interval* (describe in the following paragraph) will be shifted. If the dependency ratio remains no less than 3% when B requests A for new blocks 30 seconds later, this link keeps idle for another 30 seconds. However, if the dependency ratio is greater than 5%, the system will tear down this logical link.

Encoding interval: In systems with complete random linear network coding (e.g., Avalanche), all blocks at a peer are encoded. This results in fast information diffusion to corresponding receivers. With sparse coding, all blocks at a peer are with the same probability to be encoded. So some newly incoming blocks at a peer may not be selected while encoding, and the diffusions of these blocks in the network are slowed down in some cases. Let us use an example to illustrate this concept. Let A and B be two end peers of a link. Assume that A holds blocks $B_1^{'}, B_2^{'}, \ldots, B_q^{'}$ at time t, and receives a block $B_{q+1}^{'}$ at time t_1. With sparse encoding, candidates of encoded blocks at A range from $B_1^{'}$ to $B_q^{'}$ before t_1 and extend to $B_{q+1}^{'}$ afterwards. However, $B_{q+1}^{'}$ may not be selected while encoding, because it only has a probability of $(\log n + d)/(q + 1)$ to be selected. To deal with this problem, a so-called *encoding interval* technique is used in our system. In the following period after time t, B will firstly request A for new blocks from $B_1^{'}, B_2^{'}, \ldots, B_q^{'}$, except for the following three cases: (1) B receives two dependent blocks continuously from A. (2) B has received no less than 3% linearly dependent blocks from A. (3) B has already received q independent blocks from A. In the above three cases, if A holds more than q blocks $B_1^{'}, B_2^{'}, \ldots, B_q^{'}, B_{q+1}^{'}, \ldots, B_{q+m}^{'}$ for $m \geq s (s = \log n + 10)$, *encoding interval* at A is shifted to $B_{q+1}^{'}, B_{q+2}^{'}, \ldots, B_{q+m}^{'}$, i.e., the encoding candidates at A range from $B_{q+1}^{'}$ to $B_{q+m}^{'}$ at the instant. If $m < s$, B will stop requesting A for new blocks for 30 seconds. Notice that if the *encoding interval* is used in the system, the probability of a block being encoded is $p = (\log n + d)/k^{'}$, instead of $p = (\log n + d)/k$, where $k^{'}$ is the length of *encoding interval*.

III. Experimental Evaluation

In this section, we present the experimental results and discuss some interesting findings from the experiments.

A. Encoding and Decoding Rate

When it comes to sparse linear coding, one may mostly care about its encoding and decoding rate, versus complete linear coding. We carry out the tests of encoding and decoding rate on a single PC machine (Intel Celeron Northwood Processor 2.0GHz, 512M memory) with a 100MB file as the original content. In the tests, the encoding density is set to $p = (\log n + 10)/k$ for sparse coding, and the block size varies from 64KB to 2048KB, where n is the number of total blocks and k is the number of blocks held at a peer. Fig. 2 illustrates the comparison of encoding rates between sparse coding and complete linear coding with different block sizes. Experimental results show that sparse coding greatly increases the encoding rate and can reach 4MB/s in some cases. Furthermore, one can achieve such a high encoding rate without loading the whole 100MB file into the memory during the encoding process. It is clear that sparse encoding rate can meet the common transmission bandwidth requirement for content distribution even without code optimization. On the other hand, complete linear coding reaches encoding rate of only 300KB/s for block size 256KB, less than 10% of sparse coding. In a content distribution session, the density of the coefficient matrix is very likely to increase since the received blocks at a peer are possibly multiple encoded. In the

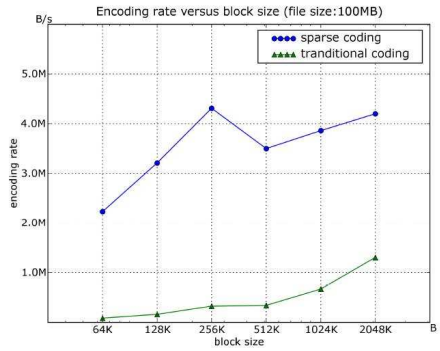

Fig. 2. Encoding rates(B/s) of sparse coding and complete coding

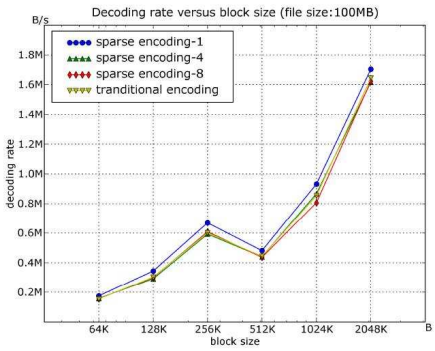

Fig. 3. Decoding rates(B/s) of multiply sparse coding and complete coding

test of decoding rate, we execute the decoding after encoding the original blocks with 1, 4 and 8 times with sparse coding respectively. Fig. 3 illustrates the comparison of decoding rates between multiple sparse encodings and complete linear coding. Decoding rates are very low compared with encoding rates as shown in Fig. 2. From Fig. 3, one can observe that sparse coding and complete coding achieve almost the same decoding rate for various block sizes. Although the coefficient matrix tends to be dense after multiple encodings, the decoding rates are almost the same for different times that the blocks are encoded. This is because if a matrix is sparse, its inverse is usually not a sparse matrix. Another point to note is that sparse coding helps very little in the decoding process with Gaussian Elimination. From Fig. 2 and 3, coding and decoding rates increase as the block size grows. For a content of fixed size, the number of blocks and the order of coefficient matrix decrease as the block size increases. So the decoding computational complexity decreases as the block size increases. However, decoding rates decrease for both sparse coding and complete coding in the case of block size varying from 256KB to 512KB. We believe that this phenomenon is due to the cost of memory access and cache size.

B. Encoding Density

To study the effect of encoding density on system performance, we carry out experiments for content distribution of a 128MB file on a system with 50 PC machines and the file block size is set to 256KB. We use $p = (\log n + d)/n$ as the encoding density at the source peer and $p = (\log n + d)/k^{'}$ at non-source ones, where $n = 512$ for a 128MB file with

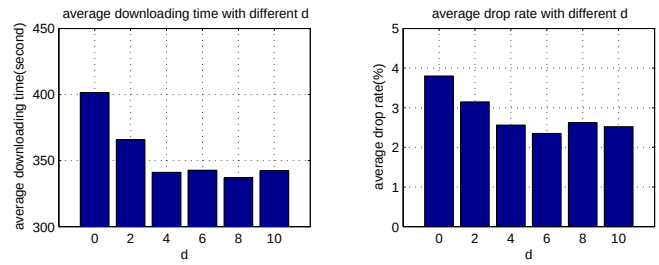

Fig. 4. System performance with different encoding density parameter *d*

block size 256KB, and $k^{'}$ is the length of the *encoding interval*. The system performance is evaluated by averaging the downloading time and drop rate at all non-source peers. Once a peer holds n independent blocks, it will stop receiving data from others.

Fig. 4 illustrates the experimental results with *d* ranging from 0 to 10. One can observe that the system performance is the worst (i.e., with the longest average downloading time) and has the largest drop date when $d = 0$, compared with $d \geq 4$. As *d* increases, the system performance improves, till $d = 4$. For those cases that $d \geq 4$, the average downloading time and average drop rate are almost the same for different values of *d*. we conclude that larger *d* will lead to lower rate of linear dependency (according to Cooper's theorem). However, systems using sparse linear coding $d = 10$ and complete coding (one can regard it as a sort of sparse coding with very large *d*) achieve almost the same performance, where drop rate remains only 2% due to some specific topologies such as circles. Table I shows the average downloading times (AverDownTime), total distribution times (TotalDisTime) and drop rates of sparse coding with $d = 10$ and complete coding. Hence, we conservatively use $d = 10$ for sparse coding in our system.

TABLE I

32MB FILE DISTRIBUTION USING DIFFERENT CODING SCHEMES

	AverDownTime	TotalDisTime	DropRate
Sparse Coding(*d*=10)	302.6s	367.8s	2.087%
Complete Coding	293.8s	369.4s	1.802%

From Table I, one may doubt about the advantage of using sparse coding, as compared to complete coding. In fact, the experiments in Table I are executed for a file with 32MB in size. We also execute another file distribution of a 128MB file with complete coding. Experimental results show that complete coding of a 128MB content is extremely time-consuming that the total bandwidth of all links at the source peer is less than 100KB/s. This implies that complete linear coding has high overhead and reduces the system performance. So we use a 32MB file as the content for distribution in the experiments in Table I.

C. Encoding Intervals

We use *encoding interval* to speed up blocks diffusion and execute some experiments to study the performance with encoding interval. The experimental results are shown in Fig.

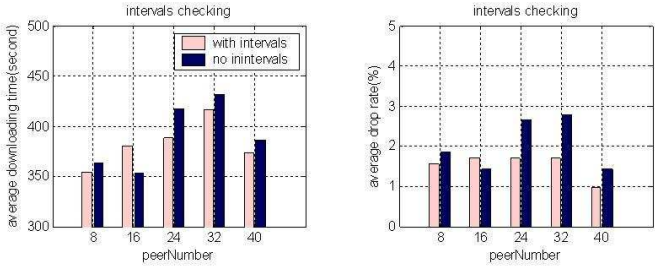

Fig. 5. Benefit of *encoding interval*

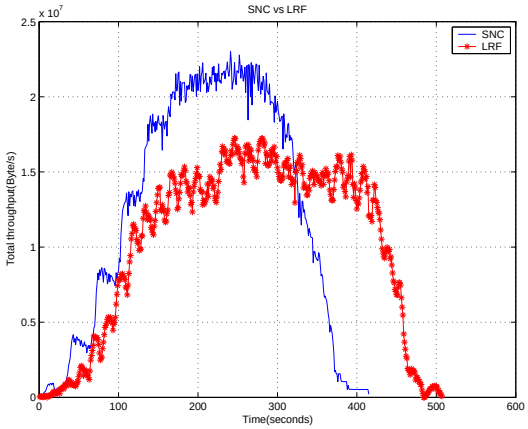

Fig. 6. Comparision of achieved throughput between sparse coding and non-coding

5. When there are at least 24 peers participating in the distributing session, the system using *encoding interval* performs much better than the system without *encoding interval* in terms of average downloading time and drop date.

However, the system with *encoding interval* performs almost the same with the system without using the *encoding interval* when only 8 peers participate in the distributing session, and even worse in the case of 16 peers. The main reasons are: (1) *Encoding interval* is mainly to reduce the probability of generating dependent blocks at a peer during the dynamic increase of blocks. With *encoding interval*, a peer will generate less dependent blocks when it receives blocks. When only few peers participate in a distributing session, two adjacent peers may have many dependent blocks with each other. *Encoding interval* does not help in this case. (2) In the case of peer A receiving two dependent blocks continuously from peer B, the link between A and B will be idle for 30 seconds if the next *encoding interval* in B is not long enough. If only few peers participate in the distributing session, it is possible that two adjacent peers have many dependent blocks with each other. This will lead to many idle links in the system and further reduce the system performance. However, it is worthwhile to note that for realistic networks, there are usually many peers participating in the file distribution.

D. Performance versus non-coding system

One important question we need to address is whether network coding can really improve the performance of P2P content distribution. With the same architecture of the system with sparse coding, we implement a P2P content distribution system without coding, in which, the *local-rarest-first* chunk selection policy is used (which is the block-selection scheme in the BitTorrent protocol). We carry out experiments and compare the performance between the system with sparse coding and the one without coding in terms of throughput, average downloading time and total distribution time. We do not include the decoding time in the total distribution time because content can be decoded offline.

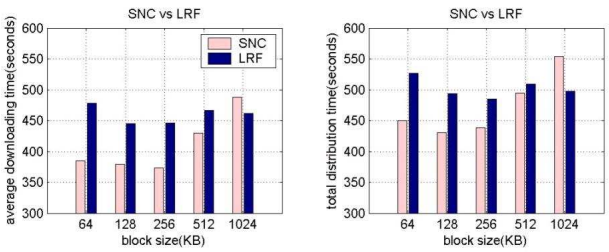

Fig. 7. Performance of distributing a 128MB content with different block sizes

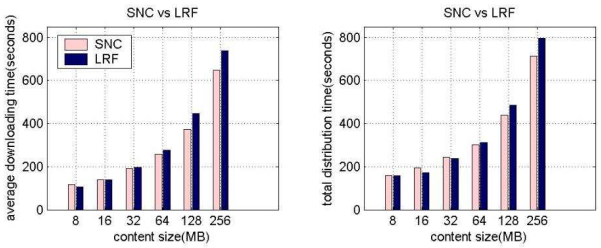

Fig. 8. Performances of distributing different sizes of contents with fixed block size

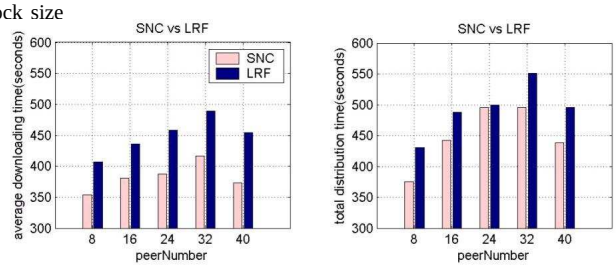

Fig. 9. SNC vs LRF in network with different scales

Throughput: Fig. 6 shows the throughputs of these two systems for distributing a 128MB content among 50 peers. One can observe that the throughput with sparse coding reaches a higher maximum throughput and also faster to the maximum throughput than the non-coding one. The reason is that it is easier for all peers to get the information of rare blocks in encoded blocks than in non-coded ones. Rare blocks are not the bottlenecks for throughput in coded system any more. So network coding speeds up the process of distributing content blocks.

The effect of block size: For a 128MB content distribution within 40 peers, we vary the block size from 64KB to 1024KB to study the effect of block size on the performance. From Fig. 7, both of the downloading time and the distribution time with sparse coding vary very little as the block size increases

from 64KB to 256KB, but the downloading time is $15-20\%$ and the distribution time is $10-15\%$ less than the system without coding. Smaller blocks for a content of fixed size incur larger number of blocks and larger order of the coefficient matrix. The overheads of coding and transmitting the blocks will be high for sparse coding with small blocks because of the following two reasons: (1) Larger order of the coefficient matrix implies higher encoding computational complexity. (2) Transmitting the coefficient matrix is more time-consuming for large order matrix. For example, the coefficient matrix is a file of 16MB for a 128MB content with block size 32KB, but only 4MB with 64KB and 1MB with 128KB. However, as the block size continues to grow, the downloading time and the distribution time of sparse coding increase. Comparing with non-coding system, both the downloading time and the distribution time of sparse coding with block size 512KB are less, but with block size 1024KB are more. While encoding, the block number will be small when the block size is large for a fixed size content. Small block number will lead to high probability of receiving dependent blocks and high drop rate. So system using network coding with too large block size performs worse than non-coding system. Moreover, if the drop rate reaches 3%, it is very likely that system performs worse as links are paused more frequently, which incurs poor transmitting efficiency among peers. Therefore, 256KB may be an appropriate block size for our system. On the contrary, the change of block size has little influence on the performance of the non-coding system.

The effect of content size: Next, we vary the size of the content from 8 MB to 256 MB and the block size is set to 256KB. From Fig. 8, the performance of the system with sparse coding is better than the non-coding system for content with size no less than 64MB, but worse for content with size no greater than 32MB. As the content size increases($\geq 64MB$), the system with sparse coding performs better than non-coding. From Table II, the drop rate is greater than 5% when content size is no greater than 32MB. This will lead to many idle links in our system and bad system performance with coding. So our system with network coding performs worse than non-coding for small size of content.

TABLE II

DROP RATE INCREASES AS THE CONTENT GOES SMALLER

FileSize(MB)	8	16	32	64	128	256
aveDropRate(%)	12.1	6.5	7.5	4.3	1.0	3.3
aveDropNumber	4.0	4.1	9.6	11.0	4.9	33.5

The effect of peer number: For a content of 128M and block size of 256KB, we carry out experiments on networks with different scales (i.e., number of peer ranges from 8 to 40). Fig. 9 illustrates the result. For all cases, the system with sparse coding performs better than the non-coding one. Especially for the case of 40 peers, sparse coding significantly outperforms non-coding. We believe that as the network scale grows, the probability of having a circle decreases, and the performance of coding system improves.

IV. CONCLUSION

We implemented a P2P content distribution system based on sparse network coding. The sparse network coding technique in the system decreases the linear dependency between encoded blocks, reduces encoding computational complexity and keeps a high probability of successful decoding. *Encoding interval* is adopted in the system to reduce the probability of transmitting dependent blocks. To prevent high dependent rate that cyclic topology brings, we test the block drop rate for each link, and pause / terminate "useless" links with *drop rate* reaching 3% / 5%. Experimental results show that the performance of content distribution with sparse coding is a little better than non-coding system using *local-rarest-first*.

The decoding time is not included in either the downloading time or the distribution time with sparse coding in this paper, because decoding can be implemented offline and the decoding time would be greatly decreased by code optimization or assembly programming. The *drop rate* in this paper can be reduced to zero by sending the coefficients of a block first, and then delivering its body only if its coefficients indicate that it is innovative. However, this strategy needs one more handshake for delivering every block. Its benefit may be counteracted in the case of low *drop rate*, e.g. less than 5% in our system.

In the future, we will research the effect of separating the coefficients and data in our system, the benefit of sparse coding for robustness versus non-coding system, and the performance of our system in the PlanetLab.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their meaningful revision suggestions.

REFERENCES

[1] R. Ahlswede, N. Cai, S. -Y. R. Li and R. W. Yeung. "Network information flow" *IEEE Trans. on Information Theory*, vol. 46, pp. 1204-1216, 2000.

[2] T. Ho, R. Koetter, M. Medard, D. Karger, and M.Effros, "The Benefits of Coding over Routing in a Randomized Setting," in *Proc.ISIT2003*, 2003

[3] P. A. Chou, Y. Wu and K. Jain,"Practical Network Coding", *Allerton Conference on Communication, Control and Computing*, Monticello, IL, October 2003

[4] C. Gkantsidis and P. Rodriguez. "Network coding for large scale content distribution", In *Proc. INFOCOM*, 2005.

[5] Mea Wang, Baochun Li,"How practical is network coding?" in *Fourteenth IEEE international Workshop on Quality of Service 2006*, Yale University, New Haven, Connecticut, Jun 19-21, 2006

[6] Dah Ming Chiu, Raymond W Yeung, Jiaqing Huang and Bin Fan "Can Network Coding help in P2P networks" Invited paper in *Second Workshop of Network Coding*, in conjunction with WiOpt, April

[7] C. Cooper. "On the distribution of rank of a random matrix over a finite field", *Random Struct. Algorithms* 17(3-4): 197-212 (2000)

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", in *ACM SIGCOMM* 2001