Network Coding for Large Scale Content Distribution

Christos Gkantsidis and Pablo Rodriguez Rodriguez
Microsoft Research
Cambridge, CB3 0FD, UK
Email: gantsich@cc.gatech.edu, pablo@microsoft.com

***Abstract*— We propose a new scheme for content distribution of large files that is based on network coding. With network coding, each node of the distribution network is able to generate and transmit encoded blocks of information. The randomization introduced by the coding process eases the scheduling of block propagation, and, thus, makes the distribution more efficient. This is particularly important in large unstructured overlay networks, where the nodes need to make decisions based on local information only. We compare network coding to other schemes that transmit unencoded information (i.e. blocks of the original file) and, also, to schemes in which only the source is allowed to generate and transmit encoded packets.**

We study the performance of network coding in heterogeneous networks with dynamic node arrival and departure patterns, clustered topologies, and when incentive mechanisms to discourage free-riding are in place. We demonstrate through simulations of scenarios of practical interest that the expected file download time improves by more than 20-30% with network coding compared to coding at the server only and, by more than 2-3 times compared to sending unencoded information. Moreover, we show that network coding improves the robustness of the system and is able to smoothly handle extreme situations where the server and nodes departure the system.

Keywords: System design, simulations, content distribution networks, collaborative networks, network coding.

I. Introduction

Up until recently, content distribution solutions relied on placing dedicated equipment at certain places inside or at the edge of the Internet. The best example of such solutions is Akamai [4], which runs several tens of thousands of servers all over the world. However, in recent years, a new paradigm for Content Distribution has emerged based on a fully distributed architecture where commodity PCs are used to form a cooperative network and share their resources (storage, CPU, bandwidth).

By capitalizing the bandwidth of end-systems, cooperative architectures offer great potential for addressing some of the most challenging issue of today's Internet: the cost-effective distribution of bandwidth-intensive content to thousands of simultaneous users both Internet-wide and in private networks, and the resilience to "flash crowds"- a huge and sudden surge of traffic that usually leads to the collapse of the affected server.

Content distribution solutions based on end-system cooperation are inherently self scalable, in that the bandwidth capacity of the system increases as more nodes arrive: each new node requests service from, but also provides service to, the other nodes. The network can thus spontaneously adapt to the demand by taking advantage of the resources provided by every end-node. The system's capacity grows at the same rate as the demand, creating limitless scalability for a fixed cost. As such, end-system cooperative solutions can be used to efficiently and quickly deliver software updates, critical patches, videos, and other large files to a very large number of users.

The best example of an end-system cooperative architecture is the BitTorrent system. BitTorrent became extremely popular as a way of delivering the Linux distributions. To enable multiple end-systems to cooperate, BitTorrent splits large files into small blocks, which allows users to download multiple blocks in parallel from different nodes. Once a user has downloaded a given block, that person's computer can immediately behave as a server for that particular block and serve anyone else looking for the file. For a detailed description of the BitTorrent system see [16].

Despite their enormous potential and popularity, existing end-system cooperative schemes such as BitTorrent, suffer from a number of inefficiencies which decrease their overall performance. Such inefficiencies are more pronounced for large and heterogeneous populations, during flash crowds, in environments with high churn, or when cooperative incentive mechanisms are in place. In this paper we propose a new end-system cooperative solution that uses *network coding*, i.e. encoding at the interior nodes of the network, to overcome most of these problems.

A. Network Coding

Network coding is a novel mechanism proposed in the last years to improve the throughput utilization of a given network topology [8]. The principle behind network coding is to allow intermediate nodes to encode packets. Compared to other traditional approaches (e.g. building multicast trees), network coding makes optimal use of the available network resources and computing a scheduling scheme that achieves such rate is computationally easy. An overview of network coding and a discussion of possible Internet applications was given in [3].

Using network coding as a building block, we provide an end-system content distribution solution which optimally uses the resources of the network. Every time a client needs to send a packet to another client, the source client generates and sends a linear combination of all the information available to it (similarly to XORing multiple packets). After clients receive enough linearly independent combinations of packets, they can reconstruct the original information.

In a big scale distributed cooperative system such as BitTorrent, finding the proper scheduling of information across the overlay topology so that nodes do not have to wait unnecessarily for new content to arrive is very difficult. This is especially the case in practical systems that cannot rely on a central scheduler and are based on local node decisions. The scheduling problem becomes increasingly difficult as the number of nodes in the overlay increases, when nodes are at different stages in their downloads, and when incentive mechanisms are introduced to prevent leeching clients. As we will see in this paper, network coding makes efficient propagation of information in a large scale distributed system with no central scheduler easier, even in the scenarios described above.

To illustrate how network coding improves the propagation of information without a global coordinated scheduler we consider the following (simple) example. In Figure 1 assume that Node A has received from the source packets 1 and 2. If network coding is not used, then, Node B can download either packet 1 or packet 2 from A with the same probability. At the same time that Node B downloads a packet from A, Node C independently downloads packet 1. If Node B decides to retrieve packet 1 from A, then both Nodes B and C will have the same packet 1 and, the link between them can not be used.

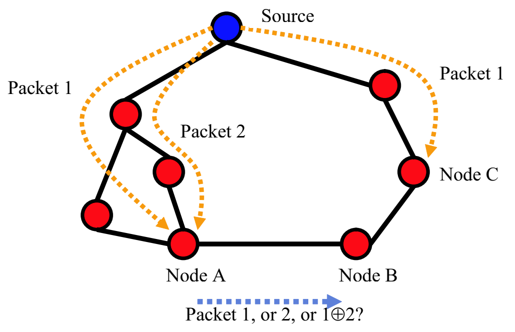

Fig. 1. Network Coding benefits when nodes only have local information.

If network coding is used, Node B will download a linear combination of packets 1 and 2 from A, which in turn can be used with Node C. Obviously, Node B could have downloaded packet 2 from A and then use efficiently the link with C, however, without any knowledge of the transfers in the rest of the network (which is difficult to achieve in a large, complex, and distributed environment), Node B cannot determine which is the right packet to download. On the other hand, such a task becomes trivial using network coding. It is important to note that the decision on which packets to generate and send at given node does not require for nodes to keep information about what the other nodes in the network are doing, or how the information should propagate in the network, thus, greatly simplifying the content distribution effort.

B. Contributions

We now summarize the main contributions of this paper:

1) We describe a practical system based on network coding for file distribution to a large number of cooperative users. Our approach does not require knowledge of the underlying network topology. Moreover, the nodes make decisions of how to propagate packets based on local information only. By using network coding, the problem of scheduling the block propagation across a large scale distributed setting becomes much easier.

2) We provide experimental evidence in many situations of practical interest that suggests that network coding performs better than transmitting unencoded blocks or using techniques that are based on erasure codes, which can be thought as coding but only at the server. Network coding performs better by almost a factor of two compared to performing encoding at the server and by a factor of three compared to not coding at all when the topology is clustered. Similarly, network coding

improves the download rates by almost 20% compared to source coding and by more than 30% compared to no coding in an heterogeneous network. During the early stages of a flash crowd, network coding outperforms source coding and no coding by 40% and 200% respectively. Even when the system reaches a steady-state, network coding still provides significant benefits compared to using other techniques. Moreover, when tit-for-tat incentives are used the performance of network coding is bearly impacted, while, other schemes suffer significantly.

3) We also show that our network coding system is very robust to extreme situations with sudden server and node departures. Network coding nodes are able to make progress and finish a download even if the server leaves shortly after uploading only one copy of the file to the system and nodes depart immediately after they finish their download. Without network coding, if both the server and the peers suddenly depart the system, some blocks of the original file or of the source-encoded file will disappear and nodes will not be able to finish their downloads. This demonstrates that network coding nodes require very little support from the origin server and are able to efficiently feed information to each other even under extreme circumstances.

The rest of the paper is organized as follows. In Section II we provide an overview of related work. In Section III we describe our model for end-system cooperative content distribution and discuss how network coding can be used as a building block of such a system. In Section IV we provide experimental evidence of the benefits of using network coding over sending blocks of the original file and over source coding. We summarize in Section V and discuss some open problems.

II. Related Work

Implementing large scale content distribution using end-system cooperation techniques has been the focus of multiple research efforts during the recent years.

a) Tree-Based Cooperative Systems: Two of the first proposals to use end-systems for content distribution were Yoid and EndSystem multicast, which provided an end-system overlay multicast architecture for the delivery of video and audio content [13] [14]. Following a similar concept, SplitStream [9] builds multiple overlay multicast trees to enable a more robust and fair system where all nodes contribute roughly the same to the distribution effort and the departure of a node causes minor disruptions. Similarly, Coopnet [10] forms either random or deterministic node-disjoint trees, and it includes a Multiple Description Coding layer (MDC) [15] to maximize the streaming quality experienced by the users.

Creating and maintaining shortest-path multicast trees provides an optimized architecture for the delivery of real-time streaming content. However, architectures that employ tree topologies are bandwidth-limited in that the transfer rate to a client will only be as fast as the throughput of the bottleneck link on the path from the server, and moreover, "perpendicular" connections among nodes are often hard to exploit. For file downloads, optimizing bandwidth is often more critical than optimizing delay, and therefore, tree-based architectures may not always be the best approach.

b) Mesh Cooperative Architectures: As an alternative to tree-based systems, a number of mesh architectures have also been suggested. Mesh cooperative architectures can substantially benefit from additional connections between end-systems, thus, maximizing download rates. The improvement is possible due to intelligent collaboration among peers, which efficiently use of the extra available bandwidth. Assuming that a given pair of end systems has not received exactly the same content, "perpendicular" bandwidth can be used to fill in the differences in received content, thus reducing the total transfer time. By harnessing the power of *parallel-downloads* resulting from connection to multiple nodes concurrently [18][19], cooperative architectures provide drastic performance benefits by taking advantage of nodes whose working sets are complementary.

However, one major drawback of mesh cooperative architectures is that since no trees are constructed, there is no pre-determined distribution path for the content and the mesh topology may include multiple cycles and out-of-order content delivery. In addition, nodes in the cooperative architecture often need to make local decisions which result in a non-optimal scheduling. The end result is that nodes need to wait unnecessarily for new data to arrive.

The most popular of such cooperative architectures is BitTorrent [16]. A detailed analysis of BitTorrent's performance can be found in [20] [21]. BitTorrent provides an end-system cooperative architecture to facilitate fast downloads of popular files.

To improve the efficient propagation of content among nodes, BitTorrent uses a rarest-first block download policy. Such policy attempts a uniform distribution of pieces among the nodes to prevent users who have all but a few of the pieces from waiting too long to finish their download. However, when nodes are close to finishing their download, nodes may attempt to obtain it from the server, causing unnecessary server overloading. To overcome such problem, Slurpie [17] proposes a randomized back off strategy combined with an effective group size estimator that precisely controls the load on the server.

c) Erasure Codes: A number of cooperative architectures [12] [11] have proposed the use of Erasure Codes[1] [22][23] (e.g. Digital Fountain) to efficiently transfer bulk data. The digital fountain approach enables end-hosts to efficiently reconstruct the original content of size *n* from roughly a subset of any *n* symbols from a large universe of encoded symbols. However, since the sets of symbols acquired by nodes are likely to overlap substantially, care must be taken to enable them to collaborate effectively. This makes cooperation and reconciliation among nodes more difficult than when no content is encoded. To overcome such problem [11] proposes techniques to efficiently reconciliate encoded content among different nodes using sketches, bloom filters, etc.

While encoded content can efficiently handle losses, asynchronous arrivals, and churn, locating missing data items may still be a challenge. To address the issue of efficiently locating useful encoded blocks within the system, Bullet [12] proposes a mechanism that periodically disseminates summaries of data sets uniformly sampled over a random subset of global participants.

d) Network Coding: Network coding was first considered in the pioneering work by Alswede et al. [8], where they showed that a sender can communicate information to a set of receivers at the broadcast capacity of the network provided one allows network coding.

High utilization of a given topology can also be achieved using multiple edge-disjoint distribution trees (specially in the case where all nodes are receivers). In fact, several schemes (e.g. SplitStream, CoopNet) have proposed to utilize multiple multicast trees (forest) to deliver striped data from the source to all receivers. These proposals can indeed improve end-to-end throughput beyond that of a single tree, however, computing the strategies to achieve optimal throughput using multiple trees has been shown to be NP-complete and APX-hard [24][25][26]. Instead, recent studies have shown that network coding can significantly facilitate the design of efficient algorithms to compute and achieve such optimal throughput [7] [2].

Most of the previous work on network coding is largely based on theoretical calculations that assume a detailed knowledge of the topology, and a centralized knowledge point for computing the distribution scheme. However, little effort has been made to study the practical aspects of implementing network coding on a real distributed setting. In this regard [5] considers the feasibility of applying the theoretical insights in network coding to increasing throughput in actual multicast systems over wide-area networks. In [6], Chou et al. propose a practical network coding system for streaming content. Similarly, in [2], K. Jain et al. provide (contemporaneously with this work) analytical evidence that supports the use of network coding in peer-to-peer networks. Based on the work presented in [6] and [2], we propose a practical end-system cooperative architecture that uses network coding to enable efficient large scale content distribution.

Network coding can be seen as an extension or generalization of the Digital Fountain approach since both the server and the end-system nodes perform information encoding. Note, however, that restricting erasure codes only to the origin server implies that intermediate nodes can only copy and forward packets. This results in *the same* erasure code being copied over from one node to another. Since mesh architectures contain cycles, it is possible that multiple copies of the same block arrive at a given receiver through different paths, thus, decreasing the effective capacity of the system. With Network Coding, on the other hand, a given block is combined with other *informative* blocks as it propagates through the network, thus, significantly reducing the probability of duplicate information arriving at the same receiver from different paths.

When the total number of different erasure codes in the system is very high, then, the probability of duplicate information arriving at the same receiver can also become quite small. However, for flash crowd arrivals where no node has the content, when nodes leave the system soon after they finish the download, or when the server leaves the system after serving few copies of the file, using erasure codes has far less clear benefits compared to not performing encoding at all.

III. Model

In this section, we describe our model for end-system cooperative content distribution. This model can be used to either distribute blocks of the original file, or blocks of encoded information, where the encoding can happen either only at the source, or both at the source and at the network. We will outline the basic operation of this system, emphasizing some algorithmic parameters that affect its performance. However, a detailed protocol implementation is outside of the scope of this paper. These algorithmic parameters and their impact will be studied in Section IV.

[1]we use the terms Erasure Codes, FEC, and Source Coding interchangeably across the paper

A. Model of a collaborative content distribution network

We assume a population of users[2] that are interested in retrieving a file, which originally exists in a single server. The capacity of the server is limited (a server could be an end-user) and, thus, users contribute their bandwidth resources to help other users. Since the server does not have the capacity to serve all users simultaneously, it divides the file into k blocks and uploads blocks at random to different clients. The clients collaborate with each other to assemble all the k blocks to reconstruct the original file. This is very similar to how current end-cooperative systems, especially BitTorrent, work.

We assume that users do not know the identities of all other users; they only know the identities of a small subset of them, which we call the neighborhood. We assume that the neighboring relation is symmetric, i.e. if node A is in the neighborhood of B, then, also, B is in the neighborhood of A. Each node can exchange information, which includes blocks of the file and other protocol messages, only with its neighbors. The size of this subset is normally a small value (e.g. 4-6).

The way nodes join the network is as follows. Upon arrival, each user will contact a centralized server that will provide a random subset of other users already in the system (similar to the tracker concept in [16]). The new user will then connect to each of them to construct its neighborhood. The end results is a mesh overlay topology where information flows along the edge-nodes in that topology. Rather than using a centralized server, other mechanisms for providing random subsets of nodes can be used like the ones proposed in [28] and [1].

In the case that some nodes loose some of their neighbors (because they left the system), or when a node needs to use more neighbors to improve its parallel download rate, the node can request additional neighbors at any time. Thus, allowing the topology to reconfigure when needed.

In this work, we assume that the major bottleneck in the system is the capacity of the access link of each user (and of the server). The total rate by which a user can receive information from all its neighbors is limited by the download capacity of the user; similarly, the total rate by which the user can upload information to all its neighbors is limited by the upload capacity of the user. For the purpose of this paper we assume symmetric links, where the download capacity is equal to the upload capacity of a node and both capacities are independent. We have experimented with asymmetric access capacities and observed very similar results and thus we omit the details of this case.

[2]we use the terms nodes and users interchangeably across the paper

B. Content propagation of uncoded and source-encoded information

Each time there is a transfer of a block from the server or a user to another user, a decision process is involved as to which block will be downloaded. We assume that neither the server, nor any user have a complete information about the blocks that each user in the system has. Instead, each user only knows about the blocks it has downloaded and the blocks that exist in its neighbors and, thus, the algorithm for deciding which block to transfer is based on local information only. In our system, we have experimented with the following heuristics, which are commonly used in current systems:

- *Random block.* The block to be transfered is decided at random among the blocks that exist in the source (if the source is the server, then a random block among all blocks).
- *Local Rarest.* The block to be transfered is picked among the rarest block in the neighborhood. If there are multiple rarest blocks, a block at random is picked among them.
- *Global Rarest.* This is a baseline scheme which is not practical in large networks. The block to be transfered is the system-global rarest block that exists in the neighborhood. This is a heuristic that gives priority to very rare blocks in the hope of improving the performance of the system.

The BitTorrent system uses a combination of the *Random* and *Local Rarest* schemes. In the beginning each nodes uses *Random* and after a few blocks have been downloaded it switches to *Local Rarest*.

When server coding is used, the system works very similar to the description above. However, the server gives blocks of encoded information and not of the original file. If the server uses forward error correction codes, then it generates $k \cdot e$ blocks, where k is the number of blocks in the unencoded file and $e > 1$ is the expansion factor, a parameter decided by the server. If the server uses rateless codes, then each time the server needs to upload a block to a user, a new encoded block is generated.

C. Content propagation with network coding.

In the case of network coding, both the server and the users perform encoding operations. Whenever a node or the server needs to forward a block to another node, it produces a linear combination of all the blocks it currently stores. The operation of the system is best described in the example of Figure 2.

Assume that initially all users are empty and that user A contacts the server to get a block. The server will

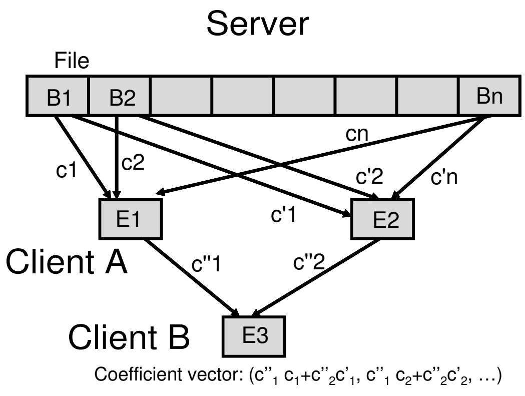
Server
File
B1
B2
Bn
c1
c2
cn
c'2
c'n
c'1
E1
E2
Client A
c"1
c"2
Client B
E3
Coefficient vector: (c"$_1$ c$_1$+c"$_2$c'$_1$, c"$_1$ c$_2$+c"$_2$c'$_2$, ...)

Fig. 2. Sample description of our network coding system.

combine all the blocks of the file to create an encoded block $E1$ as follows. First, it will pick some random coefficients $c_1, c_2, \ldots, c_n$, then multiply each element of block i with c_i, and finally add the results of the multiplications together. All these operations take place in a finite field. Observe that the probability that two independent nodes with the same set of blocks pick the same set of coefficients and construct the same block depends on the size of the field. If the field is very small such "collisions" may happen and they will reduce the performance of the system. In most practical cases a field size of 2^{16} should be enough. (For a related discussion of the size of the field and the probability of decoding in a randomized setting see [27].)

The server will then transmit to user A the result of the addition and the *coefficient vector* $\vec{c} = (c_i)$. Assume now that user A has also another block of encoded information $E2$, either directly from the server or from another peer, with its associated vector of coefficients. If user A needs to transmit an encoded block $E3$ to user B, A generates a linear combination of its two blocks $E1$ and $E2$ as follows. User A picks two random coefficients c_1'' and c_2'', multiplies each element of block E_1 with the coefficient c_1'' and similarly for the second block E_2, and adds the results of the multiplication. The block transmitted to user B will be the addition of the multiplications $c_1'' \cdot E_1$ and $c_2'' \cdot E_2$. The coefficient vector $\vec{c''}$ associated with the new block is equal to $c_1'' \cdot \vec{c} + c_2'' \cdot \vec{c'}$.

Observe that a node can recover the original file after receiving k blocks for which the associated coefficient vectors are *linearly independent* to each other. The reconstruction process is similar to solving a system of linear equations.

The benefit we expect to get by using network coding is due to the randomization introduced each time we generate a new encoded block. Recall that without network coding, each user needs to decide which block to receive based on a local decision. This may be a suboptimal decision since such block may be already well represented in other areas of the network. On the other hand, with network coding, we perform a linear combination of all available blocks at a given node. Both popular as well as unpopular blocks are combined into a single block without having to estimate their popularity. If at least one of the combined blocks is of use to other nodes down the path, then the linear combination will also be useful. The only way that a generated encoded block is not useful is if the same block was generated independently elsewhere in the system (or from the same node in a previous transmission). However, as blocks traverse the network they get combined with other blocks in different nodes, making the probability of that happening particularly small.

Observe that in the case of network coding we do not have to worry about how to pick the block to transmit to the other node; we combine all the available blocks. However, deciding whether a neighboring node can send an *innovative* packet is more difficult since the coefficient vectors of the sender can be different to the vectors of the receiver, but, still, span the same space. A simple approach is to ensure that each node knows the coefficient vectors of its neighbors. By using the neighbors' coefficients and its own coefficients, a given node can easily calculate the rank of the combined matrices and determine which nodes can provide new blocks and moreover how many blocks they can provide.

An alternative and cheaper approach is to have the sender generate a linear combination with random coefficients of the coefficient vectors available to it and send the resulting coefficient vector. If the receiver determines that the received vector is a linear combination of the vectors already available at the receiver, then it assumes that the sender does not have innovative blocks to send and waits for future updates from the sender. Observe that an unlucky choice of the random coefficients may lead the receiver to conclude that the server does not have innovative information, when in reality it does; such unlucky events should be very rare.

Note that the overhead of transmitting the coefficient vectors is quite small. In most practical scenarios, the size of each block is normally in the order of several hundreds of KBytes [18] whereas the size of a coefficient vector is smaller than one packet.

D. Incentive Mechanisms

An important problem of current collaborative content distribution networks is free-riding; many users take advantage of the resources offered to the network by other users without contributing their own resources. Free-riding can seriously degrade the performance of the content distribution [30], and, as a result, many collaborative networks have introduced mechanisms to discourage free-riding.

In our system we use two mechanisms to discourage free riding. The first is that we give priority to exchanges over free uploading to other nodes. In other words, when there is contention for the upload capacity of a user, the user will preferentially upload blocks of information to neighbors from which it is also downloading blocks. Thus, the nodes allocate their capacity preferentially to mutual exchanges and then use the remaining upload capacity for free downloads.

The second incentive mechanism that we use is inspired by the tit-for-tat approach used in the BitTorrent network [16]. A user does not upload content to another user unless it has also received enough content from that user; more specifically, the absolute difference of uploading minus downloading from one user to another is bounded.

The introduction of such an incentive mechanism makes scheduling of information across a large distributed setting even more challenging. Given that nodes make decisions based on local information, a node may end-up downloading blocks that are already popular across the system and cannot be traded easily with other users. This effect gets amplified when the network frequently reconfigures. With network coding almost every block is unique and thus has higher chances of being useful to other users and being traded easily.

IV. Experimental evaluation

In this section we study the performance of an end-system cooperative architecture that uses network coding and compare it with other existing approaches. In particular we study the performance of end-system cooperative architectures for a) different types of topologies, b) heterogeneous client populations, c) dynamic node arrivals, d) sudden node and server departures, and d) incentive mechanisms.

To evaluate the performance of each scheme, we calculate the time it takes for each user to download the file. We are both concerned with the average download time, as well as the maximum, and the standard deviation of the waiting times among all clients. Another performance metric, is the overall utilization of the network, or, in other words, how fast the network can push information to the users. We measure network throughput as the total number of blocks transfered in a unit of time. This metric is also related to how much load is taken away from the server. The higher the throughput in the cooperative network, the more efficiently nodes are contributing to the download, and the lower the load in the server.

To study the performance of potentially large number of users under various settings and scenarios, we have implemented a simulator of an end-cooperative system that uses different algorithms for content distribution. Our purpose was not to construct a perfectly realistic simulation, but to demonstrate the advantages of network coding in some specific scenarios, which, we believe, are of practical importance.

Our simulator is used to compare the performance of content propagation using network coding, not coding at all, and coding only at the server. The input to the simulator is a set of nodes with constraints in their upload and download capacities, an initial overlay topology that connects these nodes, the size of the file to be distributed to the nodes, and the capacity of the single server in the system. The capacities are measured as the number of blocks that can be downloaded/uploaded in a single round. The number of blocks of the file transfered between two users is always an integral number. We have experimented with finer granularities and observed very similar results.

Whenever a user joins the system it picks four nodes at random and makes them its neighbors (provided that they have not exceeded the maximum number of neighbors, which is set to six in most of our experiments). The simulator supports dynamic user populations with nodes joining and leaving the system, and topology reconfigurations. In fact, at the end of each round, if a node determines that the utilization of its download capacity in the most recent rounds drops below a certain threshold (10% in most of our experiments), then it tries to discover and connect to new neighbors. Similarly, if the user has exceeded its maximum number of neighbors, then it will drop some of the old neighbors at random.

The simulator is round based. At the beginning of each round, each peer contacts its neighbors to discover whether there are new blocks that can be downloaded. For unencoded content and for source coding we assume that the each node knows the blocks available at its neighbors; for network coding we use the techniques described in Section III-C. Then, the simulator decides which blocks will be exchanged so that the upload and download capacities are not violated and that exchanges take priority as explained in Section III-D. Nodes with free download capacity contact the server. However, the number of nodes that can be satisfied by the server is bounded by the server's capacity. The block transfers, either from a peer or from the server, take place at the same round and, then, the system moves to the next round.

During each simulation all the nodes in the system use the same encoding scheme, either network coding, source coding, or propagating of original blocks. To simulate rateless codes we set the expansion factor e to be very large.

To simulate a tit-for-tat scenario, the simulator keeps track of the difference between uploaded blocks minus downloaded blocks from a user S (source) to a user D (destination). If the difference is larger than the pre-configured value (typically 2 when we study tit-for-tat), then the source S will not send any block to D even if there is spare upload capacity at S and spare download capacity at D.

Obviously, there are important parameters that we do not simulate such network delays, locality properties in constructing the overlay, cross-traffic impact, or malicious users. However, we believe that the simulator is able to capture some of the most important properties of an end-cooperative architecture.

Next we present the experimental results based on the simulator described above.

A. Homogeneous topologies

We start by comparing the performance of network coding (NC) to source coding (FEC) and unencoded information using a local rarest policy (LR) in a well-connected network of 200 nodes[3], where all nodes have the same access capacity equal to one block per round (homogeneous capacities). The upload capacity of the server is also 1 block/round. In this simulation we give priority to mutual exchanges, as described in Section III-D, but do not use the tit-for-tat mechanism.

In Figure 3, we plot the finish times of each node and the progress per round for that configuration. We measure the finish times as the number of rounds required to complete the download. We observe that in this baseline scenario all schemes perform equally well. The performance with network coding was slightly better compared to the other schemes, but, still, in all schemes the average finish time was close to the minimum finish time possible, which is rounds since the original file is equal to 100 blocks.

In the following sections we deviate from that baseline scenario and observe that with small changes in the configuration network coding performs better.

B. Topologies with clusters

Network coding has been shown to perform well in topologies with bad cuts. In this section we examine such

[3]We have performed limited experimentation with larger topologies of sizes up to 2000 nodes showing similar results

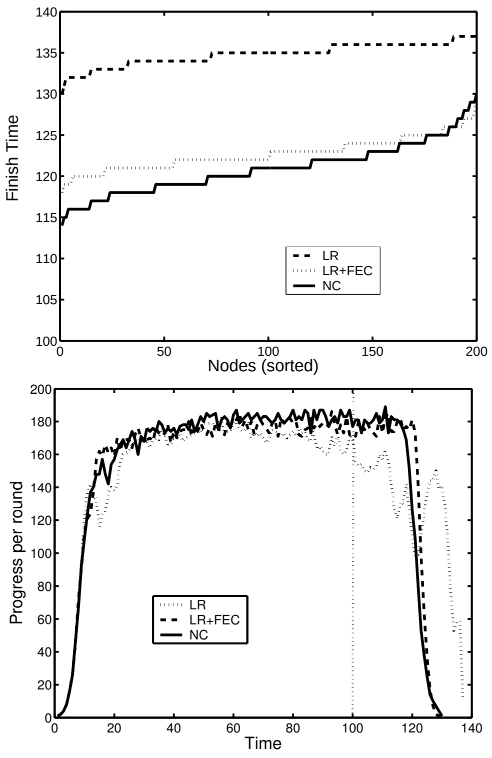

Fig. 3. Finish times and progress per round for a well-connected topology of 200 nodes. Size of the file is 100 blocks.

a topology with two clusters of 100 nodes each. There is good connectivity and ample bandwidth between the nodes in each cluster (equal to 8 blocks per round in both directions and for all nodes). However there is limited bandwidth between the nodes of the different clusters; in every round only 4 blocks can propagate from one cluster to the other. The capacity of the server is also 4 blocks per round and moreover the server departs at round 30 and, thus, the nodes need to use the capacity between the clusters in the optimal way to retrieve the blocks of the file as fast as possible.

In Figure 4 we plot the finish times for each node. The minimum possible finish time in this experiment is equal to 25 rounds. Observe that without coding the average finish time is roughly three times longer compared to using network coding. The reason is that without coding some packets are transmitted multiple times over the bad cut wasting precious capacity that could have been used to transmit new blocks. For similar reasons, network coding outperforms source coding in this example by almost a factor of two. Given that nodes stay in the system after the download is complete and the server puts 20% extra erasure codes in the system, with source coding the chances of transmitting the same block multiple times over the cut are reduced. Thus, source coding performs much better than no coding, but still

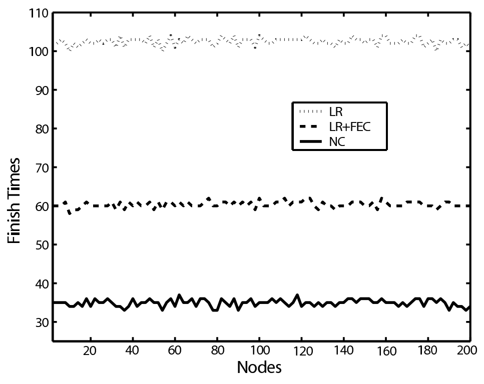

Fig. 4. Finish times of a topology with two clusters (100 nodes each).

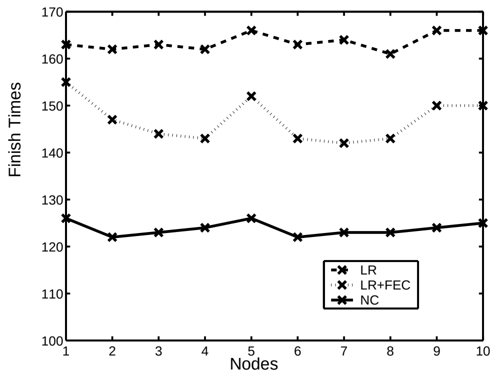

Fig. 5. Finish times of the fast nodes in a network with 10 fast nodes and 190 slow nodes. Size of the file is 400 blocks and capacity of server and fast nodes is 4 blocks/round.

worse than network coding.

In this example, for both source coding and transmitting of original packets, we have assumed that nodes use the local rarest heuristic to decide which blocks to receive from their neighbors. Choosing random blocks gave worse finish times.

C. Heterogeneous capacities

We expect that the nodes of a user collaborative distribution system will have non-homogeneous capabilities. The majority of the users are connected behind slow access links, including dialup connections and ADSL connections, and a small percentage are connected with very fast links (e.g. corporate and university users). In this regard, we wish to study the performance experienced by fast users when they interact with other slower users in such heterogeneous environments.

Since fast users have more capacity, they are allowed to have more neighbors to fully utilize their extra capacity. However, the higher the number of neighbors feeding a fast node, the harder it is for the large set of neighbors to efficiently provide useful data in an uncoordinated fashion.

In Figure 5 we plot the finish times of the fast users in a network with many slow users and few fast users. In this example the fast users are 4 times faster than the slow users. We observe that with network coding the finish times of the fast nodes are on the average 20% better than source coding and around 30% better than with no coding. Also, observe that with network coding the difference in the observed finish time minus the minimum finish time is very similar to the baseline scenario of Figure 3, indicating that the heterogeneity did not affect the fast nodes with network coding, but had decreased the performance of the fast nodes with both source coding and no coding.

TABLE I
FINISH TIMES FOR A FAST NODE AS THE RATIO OF THE CAPACITY OF THE FAST NODE OVER THE CAPACITY OF THE SLOW NODES INCREASES.

Method	x2	x4	x8
Random	107	166	281
Local Rarest	106	135	208
Source Coding Random	84	113	134
Source Coding LR	78	92	106
Global Rarest	75	92	98
Network Coding	69	72	73

Note: A ratio of x2 indicates that the capacity of the fast peer is two times the capacity of a slow peer. Similarly for x4 and x8. The number of slow peers is 50, 100, and 200 in the three cases respectively.

When network coding is not used, slow nodes may pick blocks from the server that are not needed by the fast nodes. Also slow nodes may use much of their capacity to share blocks that came from the fast nodes in the first place. The end result is that often the decisions taken by the many slow nodes do not take into account the interests of the fast nodes and fast nodes need to wait for many rounds to obtain the appropriate blocks.

On the other hand, with network coding, the blocks that propagate in the network are linear combinations of many other blocks. Thus, the fast nodes have better chances of making progress in each round.

We have also noticed that as the capacity difference between fast nodes and slow nodes increases, fast nodes experience even worse performance when network coding is not used; on the other hand, with network coding, the performance degradation is minimal. In Table I we show some results that validate it.

In this experiment we have only one fast peer and many slow peers (50, 100, and 200 for the three cases), which allows us to focus our discussion; similar results exist for the case of a small subset of fast nodes. As the ratio of the capacity of the fast peer increases from 2 to 4 and to 8, we also increase the number of neighbors of the fast node (to give it the opportunity to use the extra capacity). We also scale accordingly the capacity of the server and the size of the file so that the minimum finish time for the fast node is 50 rounds. If no network coding is used, Table I shows a drastic increase in the finish times of the fast node as the capacities ratio increases. With network coding the finish time remains relatively unchanged indicating that heterogeneity in the capacities does not penalize the fast nodes. As a final note, the performance of the slow nodes across these experiments remain almost unchanged.

D. Dynamic arrivals and departures

Dynamic Arrivals

In this section we show the impact of dynamic arrivals in the performance of the system. When nodes arrive at different times, newly arriving nodes have different download objectives than the nodes that have been in the system for sometime. For instance, newly arriving nodes can benefit from any block while older nodes require a specific set of blocks to complement the blocks that they already have. However, since newly arriving nodes do not know about the exact needs of the other nodes in the system, they will often make download decisions that are of little use to existing nodes in the system. This gets reflected in Figure 6.

In Figure 6 we simulated a scenario where 40 empty nodes arrive every 20 rounds. The file size is 100 blocks. We assume that nodes stay in the system 10 more rounds after they finish the download and the server is always available. As we can see from Figure 6, the first set of nodes that arrived at time zero finish around time 110. In the ideal scenario where existing nodes are not delayed by the arrival of new nodes, 40 nodes should finish every 20 rounds. This is clearly the case with Network Coding.

However, when no encoding is used or when source coding is used, newly arriving nodes unnecessarily delay existing nodes. Existing nodes need to wait many extra rounds to receive useful information since newly arriving nodes spend much of their bandwidth performing download decisions that carry no information for existing nodes. Such difference is amplified for the first set of arriving nodes (e.g. a flash crowd). In this situation, network coding provides an improvement of 40% and 200% compared to source coding and no coding respectively. However, as time progresses the number of nodes that finish the download and stay around to help other nodes increases. As a result, the performance difference between network coding and the other approaches

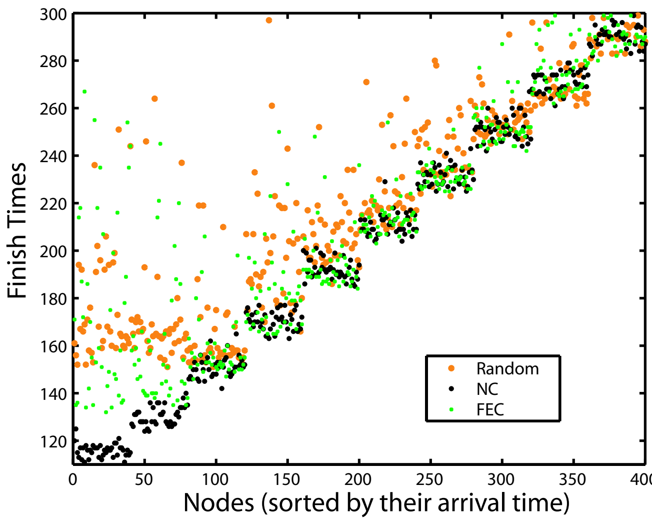

Fig. 6. Finish times for Random, Network Coding, and Source Coding (FEC) under dynamic arrivals. Nodes arrive in batches of 40 nodes every 20 rounds. Nodes stay in the system 10% extra rounds. Server stays forever. File size is 100 blocks.

decreases roughly to 30%. Note, however, that this difference would increase substantially if nodes leave the system right after they finish their download.

Robustness to node departures.

When nodes are allowed to leave the system at any time and the server can also leave the system, then it is possible that some blocks disappear and reconstructing the original file is not possible (this is can be frequently observed in current file sharing P2P networks). One possible cause that can prevent full file completion is if the block propagation does not happen efficiently and there are some rare blocks that only exist in few nodes. If the server and the few nodes holding the rarest blocks leave the system, then no node can finish the download. Such events can happen in dynamic environments. As we will see next, the inherent redundancy of network coding can help cope with such problems even in the most extreme cases of node departures.

In Figure 7 we present the finish times of nodes using network coding, with 40 nodes arriving every 20 rounds. We assume that peers leave the system immediately after downloading the complete file. We present the finish times when a) the server stays always in the system forever, and b) the server leaves immediately after uploading each block once (observe, that source coding is meaningless in this example).

From Figure 7 we can see that even in the extreme scenario where the server leaves the system immediately after distributing only one full copy of the file, network coding is able to provide the same performance as if the

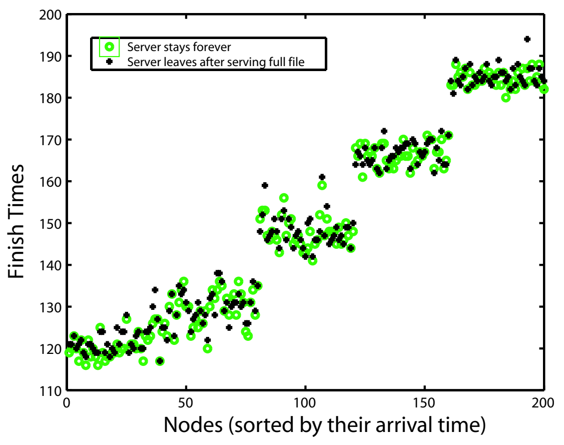

Fig. 7. Finish times for 200 nodes using network coding when a) the server stays for ever and b) when the server leaves after serving the full file. Nodes arrive in batches of 40 nodes every 20 rounds. Nodes leave immediately after downloading the file.

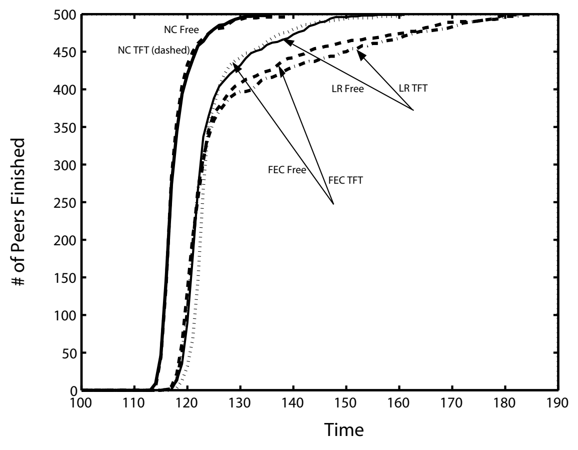

Fig. 8. Number of users finished by a given time (measured in number of rounds) for Network Coding (NC), Source Coding (FEC) and Local Rarest (LR) under a) no incentive schemes (Free) and b) tit-for-tat (TFT). Network size is 500 users. File size is 100 blocks.

server stays in the system forever. Not only the finish times are almost identical, but, also all nodes complete the download. Similarly, if we compare Figure 7 with Figure 6 where nodes stay in the system 10% extra time, we can see that the performance of network coding does not get significantly impacted even if nodes leave the system immediately after they finish the download. This results shows that nodes using network coding are very robust and self-substained, relying very little on the support of the origin server and efficiently feeding information to each other even under extreme circumstances.

If we increase the number of extra rounds that the server or the nodes stay in the system by a very small amount (for both source coding and in the case of no coding), then again we observe similar results. For instance, leaving the server for an extra 5% rounds[4], only 40% of the nodes finished downloading when source coding was used and only 10% of the nodes finished downloading when no coding was used.

However, we have experimentally observed that if the server and/or the peers stay in the system over a certain threshold (10-15% extra rounds for source coding and around 20%-30% for the case of no coding), then all users are able to finish the download, although they still may take a lot longer to receive the file.

E. Incentive mechanisms: Tit-for-tat

We have argued in Section III-D that the use of an incentive mechanism like tit-for-tat (TFT) may reduce the throughput of the system since blocks need to be carefully picked to be easily traded.

[4]Rounds above the minimum possible time to download all the blocks.

In Figure 8 we show the total number of peers finished by time t with and without the use of tit-for-tat to discourage free-riders. In this simulation the maximum allowable difference between blocks uploaded to a node minus the number of downloaded blocks from the same node is 2. In the case of network coding, the introduction of TFT has practically no observable impact on the performance of the system. However, for both source coding and no coding, the introduction of TFT significantly affects the finish times by delaying the upper tail of the distribution.

For instance, when transmitting unencoded blocks, the last user finished at time 161 without TFT and at time 185 with TFT. Similarly, when source coding was used the finish times were 159 and 182 respectively. The decrease in performance happens because nodes may end-up with blocks that are of little interest to their neighbors. Recall, however, that nodes are allowed to change neighbors if they are not able to receive enough throughput.

We have also experimented with larger networks and observed that increasing the size of the network amplifies the penalty introduced by using tit-for-tat, specially for a system where no-coding is used.

V. Summary and Further Directions

We propose a new content distribution system that uses network coding. Unlike other systems based on network coding, our approach targets the distribution of large files in a dynamic environment where nodes cooperate. Our system does not require any centralized knowledge of the network topology and nodes make decisions of how to propagate blocks of information based only on local information.

The main advantage of using network coding for distributing large files is that the scheduling of the content propagation in the overlay network is much easier. Deciding on the correct block of information to transmit to another node is difficult without global information; the transmitted packet is useful to the receiving node, but, may not be useful to other downstream nodes. With network coding, each generated block is a combination of all the blocks available to the transmitter and thus, if any of them is useful downstream, then the generated block will also be useful.

We have demonstrated through extensive simulations the performance advantages of using network coding over transmitting unencoded information and over coding at the source only in scenarios of practical interest. Network coding performs better when the nodes have heterogeneous access capacities, when the arrivals and departures of the nodes are not synchronized, when there are natural bottlenecks in the overlay topology (that need to be utilized as best as possible), and when incentive mechanisms are in place to discourage free-riders. The performance benefits provided by network coding in terms of throughput are more than 20-30% compared to coding at the server, and can be more than 2-3 times better compared to transmitting unencoded blocks. Moreover, we have observed that with network coding the system is much more robust to server and node departures.

A major concern in any content distribution scheme is the protection against malicious nodes. A malicious node can introduce arbitrary blocks in the system and make the reconstruction of the original file impossible. When the nodes do not perform coding, the server can digitally sign the packets transmitted and, thus, protect against malicious users. Digitally signing is more difficult when rateless codes are used, but recently [29] demonstrated how homomorphic collision-resistant hash functions can be used to provide protection in that case. Similar schemes can be used to provide protection when network coding is in place. We are currently investigating the applicability and the performance of using homomorphic collision-resistant hash functions to provide protection against malicious users in our system.

Despite the rich literature in network coding, we are not aware of any operational content distribution network that uses network coding. We are currently in the process of building a prototype system and study the advantages of network coding in more realistic settings. Building this prototype will also help us understand better some other practical issues with network coding, including the speed of encoding and decoding. Preliminary implementation results show, however, that network coding can be implemented with very low encoding and decoding overheads ($< 3\%$).

Acknowledgments

We would like to thank our collaborators Philip A. Chou and Kamal Jain for their invaluable insights about Network Coding which greatly inspired the design of our system

References

[1] R. Rejaie and S. Stafford, "'A Framework for Architecting Peer-to-Peer Receiver-driven Overlays"', *Nossdav 04*, Ireland, June 2004.

[2] K. Jain, L. Lovasz, and P. A. Chou, "Building scalable and robust peer-to-peer overlay networks for broadcasting using network coding", Under Submission to ACM-SIAM (SODA 05).

[3] P. A. Chou, Y. Wu, and K. Jain, "Network coding for the Internet", *IEEE Communication Theory Workshop*, Italy, May 2003.

[4] *http://www.akamai.com*

[5] Ying Zhu, Baochun Li, Jiang Guo, "Multicast with Network Coding in Application-Layer Overlay Networks," *IEEE Journal on Selected Areas in Communications*, January 2004.

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

[7] Zongpeng Li, Baochun Li, Dan Jiang, and Lap Chi Lau, "On Achieving Optimal End-to-End Throughput in Data Networks: Theoretical and Empirical Studies", *ECE Technical Report, University of Toronto*, February 2004.

[8] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow", *IEEE Transactions on Information Theory*, July 2000.

[9] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments", *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[10] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking", *Proc. of NOSSDAV 2002*, May 2002.

[11] J. Byers and J. Considine, "Informed Content Delivery Across Adaptive Overlay Networks", *Proc. of ACM SIGCOMM*, August 2002.

[12] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh", *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, 2003.

[13] Paul Francis, "Yoid: Extending the internet multicast architecture", *Unpublished paper*, April 2000.

[14] Yang-hua Chu, Sanjay G. Rao, and Hui Zhang, "A Case For End System Multicast", *Proceedings of ACM SIGMETRICS*, Santa Clara,CA, June 2000, pp 1-12.

[15] Vivek K Goyal, "Multiple Description Coding: Compression Meets the Network", *IEEE Signal Processing Magazine*, May 2001.

[16] B. Cohen, "Incentives build robustness in BitTorrent", *P2P Economics Workshop*, 2003.

[17] Rob Sherwood, Ryan Braud, Bobby Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol", *IEEE Infocom*, March 2004

[18] P. Rodriguez, E. Biersack, "Dynamic Parallel-Access to Replicated Content in the Internet", *IEEE Transactions on Networking*, August 2002

[19] John Byers, Michael Luby, and Michael Mitzenmacher, "Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads", *Infocom*, 1999.

[20] M. Izal, G. Urvoy-Keller, E.W. Biersack, P. Felber, A. Al Hamra, and L. Garces-Erice, "Dissecting BitTorrent: Five Months in a Torrent's Lifetime", *Passive and Active Measurements 2004*, April 2004.

[21] Dongyu Qiu, R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks", *To appear in Sigcomm 2004*.

[22] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data", *SIGCOMM, 1998*.

[23] Petar Maymounkov and David Mazires, "Rateless Codes and Big Downloads", *IPTPS'03*, February 2003.

[24] K. Jain, M. Mahdian, and M. R. Salavatipour, "Packing Steiner Trees", *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003.

[25] G. Robins and A. Zelikovsky, "Improved Steiner Tree Approximation in Graphs", *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2000.

[26] M. Thimm, "On The Approximability Of The Steiner Tree Problem", *Mathematical Foundations of Computer Science 2001*, Springer LNCS, 2001.

[27] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting", *ISIT*, Yokohama, Japan, 2003.

[28] G. Pandurangan, P. Raghavan and E. Upfal, "Building Low-diameter P2P Networks", *42nd Annual Symposium on Foundations of Computer Science (FOCS01)*, pp. 492-499, 2001.

[29] M. Krohn, M. FreedMan, D. Mazieres, "On-the-Fly Verification of Rateless Erasure Codes for Efficient Content Distribution", *IEEE Symposium on Security and Privacy*, Berkeley, CA, 2004.

[30] E. Adar, B. Huberman, "Free Riding on Gnutella", *First Monday*, Available at: http://www.firstmonday.dk/issues/issue5_10/adar/, 2000.