Comprehensive View of a Live Network Coding P2P System

Christos Gkantsidis
chrisgk@microsoft.com

John Miller
johnmil@microsoft.com

Pablo Rodriguez
pablo@microsoft.com

Microsoft Research
7 J J Thomson Avenue
Cambridge, CB3 0FB. UK

ABSTRACT

In this paper we present the first implementation of a P2P content distribution system that uses Network Coding. Using results from live trials with several hundred nodes, we provide a detailed performance analysis of such P2P system. In contrast to prior work, which mainly relies on monitoring P2P systems at particular locations, we are able to provide performance results from a variety of novel angles by monitoring all components in the P2P distribution.

In particular, we show that Network Coding is practical in a P2P setting since it incurs little overhead, both in terms of CPU processing and I/O activity, and it results in smooth, fast downloads, and efficient server utilization. We also study the importance of topology construction algorithms in real scenarios and study the effect of peers behind NATs and firewalls, showing that the system is surprisingly robust to large number of unreachable peers. Finally, we present performance results related to verifying network encoded blocks on-the-fly using special security primitives called Secure-Random-Checksums.

Categories and Subject Descriptors: C.2.4 [Computer - Communication Networks]: Distributed Systems-Distributed applications C.4 [Performance of Systems]: Measurement Techniques

General Terms: Measurement, Performance, Design, Security.

Keywords: peer-to-peer, content distribution, network coding, secure random chesksums, NAT issues.

1. INTRODUCTION

In recent years, a new trend has emerged with peer-to-peer (P2P) systems providing a scalable alternative for distributing commercial, legal content (e.g. [11, 19, 20]). Such systems use end-user's resources to provide a cost-effective distribution of bandwidth intensive content to thousands of users.

This paper presents our experiences with a P2P system that uses Network Coding. While our previous research showed through simulations that Network Coding provides efficient and robust distribution [11], it was believed that Network Coding is not practical in real environments because of encoding and decoding overheads, and because protecting against block corruption is difficult.

We have implemented a prototype Network Coding P2P filecasting system (described in §2), which to the best of our knowledge is the first system of its kind, and tested it in the distribution of large files (e.g. several GBytes) over the Internet. Our implementation also provides efficient support against corruption attacks that try to disrupt the download. In this paper, we present our experiences implementing and using such system.

In addition to presenting our field experiences with network coding, we also present interesting results about general P2P systems. Next, we summarize the main contributions of this paper:

a) We present the performance of a live P2P filecasting system from a novel set of angles (§4). In contrast to prior work, which mainly relied on monitoring P2P systems at particular locations, we were able to monitor all components in the P2P distribution, and, as a result, study a number of metrics that were not possible before. For example, we are able to quantify the content provider's savings over time, the dynamics of the topology, the number of unreachable nodes at any point in time, and the overall system efficiency.

b) We present our experiences with implementing and using Network Coding. We quantify the system requirements and its benefits in terms of download times. In particular, we show that coding is feasible and incurs little processing overhead at the server and the peers (§4.6). Moreover, coding is effective at eliminating the first/last-blocks problems (§4.8) and uses the server capacity very efficiently.

e) We evaluate various P2P topology construction algorithms and quantify their impact in the overall throughput of the system.

c) We study the influence of unreachable nodes (e.g. behind NATs, firewalls) in the system's efficiency (§6). We compare the percentage of unreachable nodes with the system's efficiency over time, and observe that surprisingly the system is highly resilient to large number of unreachable peers (e.g. as high as 70%).

e) We study the performance of a novel set of security functions to secure Network Coding systems, which we call *Secure Random Checksums* [12], and show that they have a negligible computational overhead and, hence, on-the-fly verification is possible (§7).

1.1 Network Coding

Network coding is a novel mechanism that promises optimal utilization of the resources of a network topology [1, 6, 18, 26]. With network coding, every transmitted packet is a linear combination of all or a subset of the packets available at the sender (similar to XORing multiple packets). Observe that encoded packets can be

IMC'06, October 25–27, 2006, Rio de Janeiro, Brazil.

further recombined to generate new linear combinations, enabling nodes to generate encoded packets without having the full file. The original information can be reconstructed after receiving enough linearly independent packets.

Network coding is of great use in large-scale distributed systems, such as multicast and wireless networks [11, 22]. In this paper we focus on the use of Network Coding for P2P networks. In contrast to the multicast scenario where the benefits of network coding relate to specific network topologies, in P2P systems the benefits of network coding mostly stem from solving the *block scheduling* problem at large scales. In particular, network coding improves performance when the number of users in the system increases while the information that each node has about others remains constant. More specifically network coding provides the following benefits:

I- The capacity of the seed server is fully utilized by constantly serving innovative information:

This is a critical point; at time zero the seed server is the only node holding a copy of the full file. Thus, the download time achieved by the earliest finishing node is lower bounded by the time that it takes to put a single copy of each block in the network. Network coding ensures that this time is optimal since every block served by the server is unique (refer to Section 4.7).

Note that in general this is hard to realize at large scales. To guarantee that an individual node does not request a block from the server that has already been requested by other nodes in the system, that individual node needs to know what the other nodes have downloaded and what they are currently downloading. However, a particular node often only knows about the content in the nodes in its neighborhood. As scale increases, the information that each node has about others in the system decreases, thus, increasing the probability of requesting overlapping blocks.

The best a node can do is to assume that other nodes in the system have a similar set (or subset) of the blocks existing in its neighborhood and request non-overlapping blocks accordingly. However (a) newly arriving nodes, (b) heterogeneous capacities, and (c) churn, can create swarms where different nodes are missing a very diverse set of blocks. This makes it harder to estimate what blocks exist in other parts of the network based on local observations.

For instance, in the extreme case where all nodes are missing different sets of blocks and the visibility in the network is very small, the amount of blocks served by the server before N different blocks are placed in the swarm tends to $NlogN$ (see [27]), which for N=10,000 blocks equals $80,000$.

II- Innovative information in a node propagates in optimal time to those nodes needing it, regardless of the number of hops between the source and the sink:

Assuming that all nodes have the same capacity, then, the speed at which information propagates over the P2P network is determined by the block selection policies and the distance between nodes. In the case that a given node (source) holds a particular block which is required in other parts of the network (sinks), network coding will ensure that such block will propagate in a number of rounds equal to the distance between the source and the sinks. However, if no coding is used (or even if only source coding is used), then, the nodes in the path may "waste" several rounds downloading blocks that the sinks already has. For instance if the nodes in the path are empty, they will likely request blocks that the sinks already posses.

Compared to traditional approaches, network coding makes optimal use of the available resources in the network without the need for sophisticated scheduling algorithms and provides a high degree of robustness, even if nodes suddenly depart the system or if decisions are based only on partial information [11]. An overview of network coding and its applications is given in [10].

2. SYSTEM OVERVIEW

2.1 Prototype Implementation

We have implemented a network coding based P2P file distribution system in C#. Our content distribution system consists of three types of participants: one or more peers, a registrar, and a logger.

Peers are sources and sinks for content data. Peers exchange encoded information with each other in units that we call *blocks*. Content is seeded into the system by a special peer, which we call *server*. Peers that finish the download, but remain in the system are called *seeds*.

The registrar enables peer discovery. The active peers periodically report to the registrar and the registrar provides a random subset of the active peers to nodes that have too few neighbors. The logger is an aggregation point for peer and registrar trace messages. Every peer in the system reports detailed statistics to the logger; using those statistics we are able to perform an in-depth evaluation of the distribution.

The peer is the most complex of the three entities, and its functionality is divided into two components: *network transport* and *content manager*. The network transport maintains connections to other peers for transferring blocks. We use two connections per pair of nodes (one for each direction). Each peer maintains 4-8 connections to other peers. Peers periodically drop a neighbor at random, encouraging cloud diversity and mitigating formation of isolated peer islands.

The content manager encodes, decodes, validates, and persists content data. In our experiments, the file is divided into 1000-2000 *original* blocks; all transfered blocks can be expressed as combinations of the original blocks. To ensure low encoding and decoding times, we have grouped blocks into so-called segments or generations [6], where only blocks of the same generation are combined. This approach, which we call Group Network Coding, results in more efficient decoding while retaining the network coding advantages. The encoding/decoding operations take place in a Galois Field (GF(2^{16})).

3. EVALUATING SWARMING SYSTEMS

P2P systems are quickly evolving into a mainstream mechanism for general content distribution. As such, they are being considered for a variety of applications ranging from small scale and large scale file sharing, to software and patch distribution, sensor data distribution, etc. Each of these applications pose different requirements on the swarming protocol (e.g. in terms of the population size, arrival patterns, content durability, available seeding resources, willingness to cooperate, etc). Evaluating swarming systems under the right context is critical to understand their benefits and limitations. We next highlight some of the key issues when evaluating swarming systems:

Swarm efficiency with scale: Nodes in swarming systems often perform local interactions with the goal to optimize the overall efficiency of the swarm. Such local actions are more or less effective depending on the visibility that a given node has on the complete system. For instance, at small scales a node's view of the system is usually very good, and local heuristics such as the "Rarest First" neighborhood policy often achieve efficiencies close to optimal. However, in a large scale systems with several thousands or millions of on-line nodes, each node only knows about the behavior of a very limited portion of the system (e.g. few tens of users), thus,

decisions tend be quasi-random. For instance, "Rarest First" algorithm will not perform well when the nodes are clustered in such a way that the perception of "rare" for a cluster is different than that of other clusters. It is thus critical to quantify whether the swarm efficiency remains high as the system scales.

Nevertheless, evaluating very large swarms (e.g. with more than several thousand concurrent users) is often a difficult task (e.g. computational requirements, lack of data, etc). An alternative approach is to test smaller swarms and reduce the visibility that each node has on the overall swarm (e.g. by decreasing the neighborhood size). This approach can be used to efficiently infer the impact of large scale deployments with small scale systems.

In our prototype application we have limited the maximum node degree to eight connections (as opposed to 40-80 used in other swarming systems [8]). We will show that even with such small number of connections, network coding can achieve very high efficiencies.

Using a small number of connections can also have interesting side benefits in different scenarios. For instance, commercial P2P systems that need to keep privacy among users, rely on encrypted SSL connections among peers. Due to the overhead of opening such connections, nodes need to minimize the number of SSL connections that they use. Similarly, modern P2P systems often use special servers to relay connections for nodes behind NATs. Each new node connection creates load on this server, which can eventually become the bottleneck of the system.

Partial observations are not enough: One common problem when evaluating swarming systems is to monitor the performance of the system from the point of view of a single or few nodes. This could result in loss of critical information information related to peers in parts of the network that are not being monitored. What happens to nodes with different speeds? How about nodes entering the system at different stages in the download? etc. Such lack of information often also prevents calculating the best possible performance of the system, and therefore, it is hard to estimate the overall efficiency.

To ensure that a swarming system is properly evaluated, it is important to observe the system from various angles and collect information at multiple points. Besides, it is critical to understand the upper bound on the capacity of the system, which can only be determined by knowing the maximum capacity of each node and their full connectivity pattern.

In our system, we have carefully placed extensive logging at all points in the system. This permits us to understand how all nodes are interacting with each other, what kind of connectivity there is among nodes, how efficiently they are using their upload capacity, or their connectivity pattern, which are all critical to determine how efficiently the system is performing.

Impact of arrival patterns: Swarming protocols have different relative impact in the system at different stages of the download. It is important to evaluate swarming protocols during the phases where they play the biggest role. One common mistake is to evaluate swarms during stable phases with many seeds or very high seed capacities. In such scenarios, most nodes are downloading the content directly from other nodes that have the full file, and block scheduling techniques across users rarely come into place.

During such stable phases, most nodes also have about the same set of blocks, and inferring what others are missing is a much easier task. However, other arrival patterns can have a much more demanding impact on the swarming protocol. For instance, if there is a continuous set of newly arriving nodes in the system, then, the set of missing blocks across nodes can be very different; while newly arriving nodes can be satisfied by any block, older nodes need very specific blocks to finish. This could create a situation where older nodes in the system do not get priority in downloading their missing blocks and they are delayed for long periods of time.

Flash crowds are among the most demanding phases in a content distribution cycle since content resides at a single node, many users interact at the same time, and there is often a large number of newly arriving nodes. Flash crowds can last few hours to several days, after which the distribution cloud moves into a more benign, stable phase with many seeds and plenty of upload capacity. Actually, for certain types of content, the stable phase is never reached. For instance, content that is highly popular and updated frequently, generates swarms with large number of users requesting the same file at once and where the file only exists at the original server. Other arrival patterns can also significantly stress the swarming system (e.g. sudden departures, nodes arriving pre-populated, repeated flash crowds, etc).

In our system, we have considered various arrival patterns, but focused on flash-crowd events where most users arrive within few hours of the file being published and the file resides at a single location with limited upload capacity.

4. RESULTS

4.1 Data Summary

Our prototype implementation was used to distribute four large data files to hundreds of users across the Internet. The total trial period included roughly four hundred clients. Clients arrived asynchronously after the notification of each trial commencement. Each individual trial only handled one single large file and trials did not overlap in time. Table 1 summarizes the data for all four files delivered. In this paper, we focus mostly on the results of Trial-4 since this posed the most stringent load requirements.

During the trial, a single server, which had an upload capacity of 2.5Mbps was used to publish the file; the same server served as registrar and logger.

Table 1: Summary of Trials.

	Trial 1	Trial 2	Trial 3	Trial 4
Duration (hours)	78	181	97	69
File Size (GB)	3.7	2.8	3.7	3.5
File Blocks	1000	2000	1000	1500
Total Clients	87	94	100	72
Bytes Sent (GB)	129.15	179.63	208.32	143.73
% from Server	33%	44%	19%	16%
% Unreachable Nodes	64%	57%	43%	40%
Avg Download Time (hr)	13	9	16	12

4.2 Individual Peers

Trial participants were diverse in terms of geographical location, access capacity and access type (e.g. corporate links, DSL/cable home users, wireless links). Figure 1 shows the user characteristics for the first trial; the slope of the line that connects point $(0, 0)$ and a user equals the average download rate of the user. Note that users were scattered across the world and while most users had download speeds between 550 Kbps and 1.6 Mbps, others connected through fast corporate links as well as slow modem connections.

It is interesting to compare the performance of a node behind a non-reachable NAT (PC NATed) and the performance of the same node (behind the same access link) without a NAT (PC non NATed). Note that the non-NATed node gets a much higher throughput than the NATed one (see Figure 1). The reason being that NATed nodes cannot be reached by other nodes, and therefore, their pairing possibilities are significantly reduced.

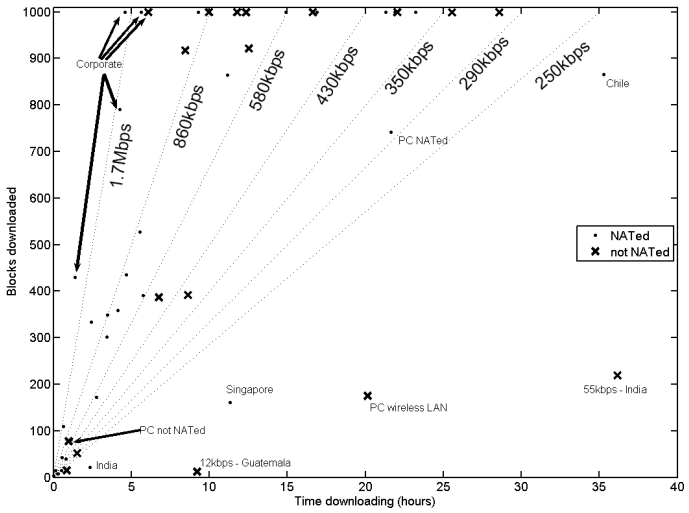

Figure 1: Summary of participating users (Trial-1). In total, there were 87 machines, out of which 31 were publicly addressable. There was a great variety of clients both in terms of geographic location, and, also, in terms of access capacities (see the dotted lines).

Figure 2 shows the set of neighbors for each peer over time in Trial 1. Observe that the set of neighbors for each peer varies from six to eight peers for downloads (except after completed) and from four to six peers for uploads for most of the trial duration.

Figure 3 also shows the number sessions for each of the 100 peers that participated in the Trial 3. We define a new session each time a user resumes its download. Observe that most peers had multiple sessions (some up to 14), with only 20% of the peers completing the download in one single session. This indicates a high level of churn in the system with users coming and going multiple times before they complete. This is an interesting observation for designining swarming systems. Such behaviour can result in many nodes entering the system with a diverse set of pre-populated blocks, which complicates the process of optimal block selection since local observations may not be representative of the content existing in other parts of the network.

4.3 System Rates

Using the detailed statistics collected in the logger, we can compute the overall system throughput, which equals the aggregate download rate, and estimate the contributions of the server, the seeds, and the clients. We plot those performance statistics for Trial-4 in Fig. 4. The total throughput of the system follows closely the total number of active users. The resources contributed by the server remained constant during the trial and the system maintained high throughput even during the beginning of the trial, where many nodes suddenly arrived and no seed nodes existed.

To better understand the system's performance, we calculate the user download efficiency. For each user, we record its average and maximum download rate, and its arrival time in the system (peers started arriving after time 10hr). The download efficiency of a user is the ratio of its average download rate over the maximum; ideally this ratio should equal 1, however the system is constrained by the upload capacities and hence lower ratios are expected. We group the nodes in three groups based on their arrival time, and we report the average download efficiency per group in Table 2. Note that during the last group interval, there was a large number of seeds

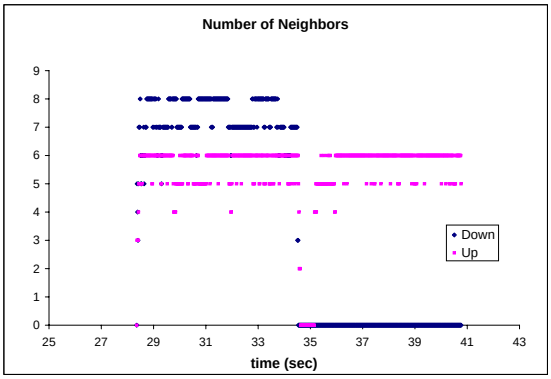

Figure 2: Number of upload and download neighbors for each peer in Trial 1.

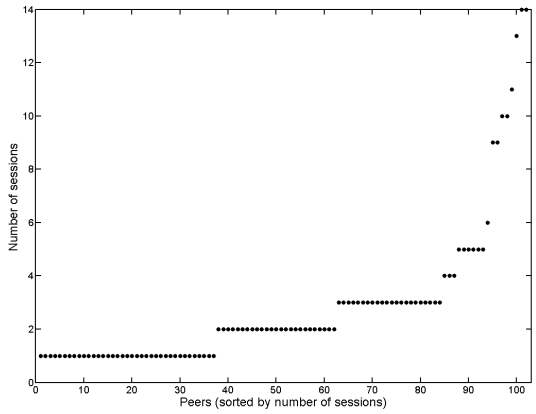

Figure 3: Number of sessions for each peer in Trial 3.

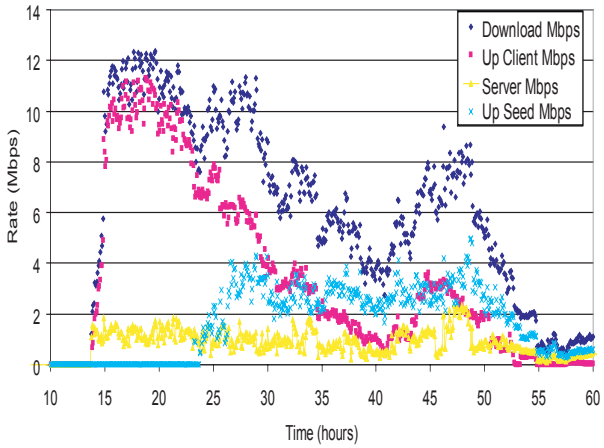

Figure 4: System Rates over time.

present, while no seeds existed during the first group interval. Observe that the efficiency is similar for all groups, including the first group, implying that nodes used the available resources efficiently even during the early stages of the trial.

4.4 Content Provider Savings

We now study the benefits of using P2P from a content provider's point of view. Recall that many hosting sites charge content owners based on the use of the egress access capacity.[1] The savings are proportional to the ratio of the aggregate download rate over the upload rate contributed by the server; the former equals the upload rate of the server in a client-server distribution. We plot that ratio in Fig. 5(left). We observe that the server saved about one order of magnitude in egress bandwidth and, hence, in monetary costs. This is a significant benefit, even for our medium sized trial, and will increase as the number of users increases.

Fig. 5(right) plots server's fair share ratio over time. To compute the fair share we divide server's upload rate by the rate that the server would have to contribute if all nodes were uploading to their maximum capacity (so that the aggregate download rate stays constant). If some nodes do not contribute with upload capacity, then, more load will be put in by the server and thus, its share would increase. Ideally the fair share should be 100% indicating that users contribute enough resources and, hence, the system could scale indefinitely. We observe that the average load on the server is $\approx 100\%$ of its fair share. The high values of fair share towards the end of the trial indicate a slightly higher usage of the server's resources, which are due to the presence of very few nodes being served mostly from the server.

[1] Often using the 95^{th} percentile of the maximum rate over a period of time.

Table 2: Average download efficiency over time

Time period	Average	St. Dev
10-20hr	0.49	0.13
20-40hr	0.5	0.16
40-60hr	0.52	0.13
Overall	0.5	0.15

4.5 Peer's Performance

We now focus on the performance seen by a typical peer. In Fig. 6 (left), we plot the actual and maximum download and upload rates for a cable modem user that has a 2.2Mbps downlink capacity and a 300Kbps uplink capacity. We observe that the average download rate is ≈ 1.4Mbps and at times reaches the maximum possible rate. The fluctuations are due to changes in the aggregate upload capacity in the system. The upload rate, on the other hand, is consistently close to its maximum value.

After the download period ended at time 34.5hr, the peer started decoding the file. Decoding finished at time 35hr, and then the peer become a seed. The upload rate increased slightly while seeding since there is no signaling in the reverse (download) direction. The zero upload rate while decoding is an artifact of the implementation and will be removed in future versions.

In Fig. 6(right) we plot the percentage of time spent by a representative sample of peers on downloading, decoding, and seeding. Observe that the time spent in decoding is less than 6% of the total download time; this time can be improved by using on-the-fly decoding and exploiting parallelization. It is also worth noting that, although some users stopped their application immediately after decoding, other stayed in the system and served other people. The average seeding time was around 42% of the total time.

4.6 Resource Consumption

We now study the resources used by our network coding implementation on a typical machine (Pentium IV @2GHz and 512MB RAM). In Fig. 7(left) we plot CPU usage during the lifetime of the user. The download period started at time 2hr and ended at time 7.2hr; during that period the CPU overhead was less than 20%. The dip in CPU's usage at time 4hr corresponds to a re-start of the application. The increase of the CPU utilization to 40% after the end of the download is due to decoding. The CPU activity droped to less than 10% after decoding and while the node was seeding.

In Fig. 7(right) we show the disk activity over the download. The spike at time ≈ 7.2hr is due to decoding. (The smaller spikes while downloading are due to activities unrelated to our P2P application.) During the experiment, we used interactive applications (e.g. word editing and WWW browsing) and did not observe any decrease in responsiveness. Overall, these results indicate that the network coding overhead in terms of end-system's resource con-

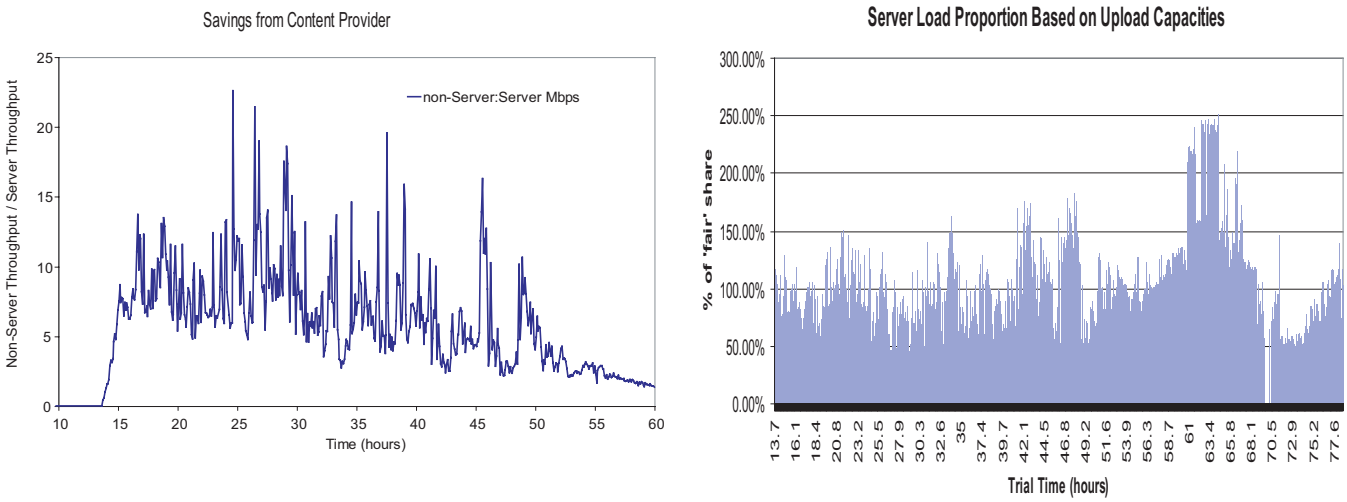

Figure 5: Content Provider effort. Left: Content Provider Savings. Right: Server Share.

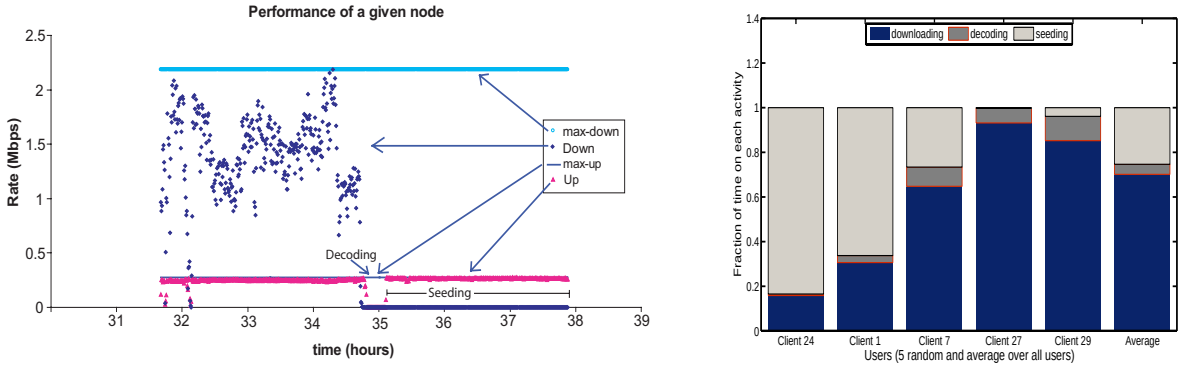

Figure 6: Description of a peer's activity and performance. Left: Peer download and upload rates. Right: Time on each activity (for 5 random peers and average).

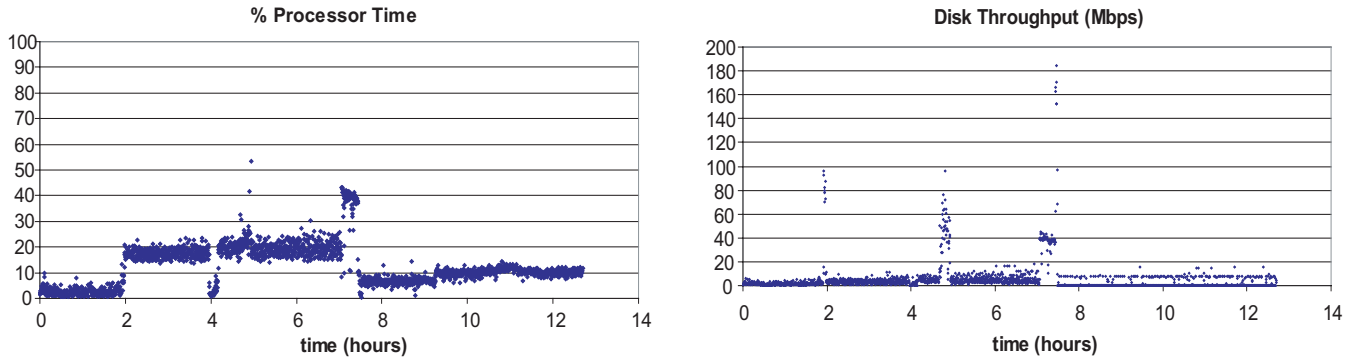

Figure 7: Resource consumption on a typical machine during Trial-4. Left: CPU Activity. Right: Disk Activity.

sumption are minimal. We expect the overheads to become negligible as we implement more sophisticated encoding and decoding techniques.

4.7 Server Efficiency

Minimizing the download time, especially of the early nodes, requires optimal utilization of the server resources. Indeed, no download can finish before the server serves each byte of the file at least once, which is equivalent to eliminating duplicate transmissions (until, of course, a full copy of the file has been served). Network coding ensures optimal resource utilization since every block generated by the server can be used (with high probability) in place of any other block to reconstruct the original file.

However, combining the entire file to construct a single encoded block is impractical due to high encoding and decoding costs. Instead, in our implementation, we divide the blocks of the file into groups (similar to the generations in [6]) and we perform coding only inside the groups. Typically, each group is composed of 50-100 blocks. By restricting the number of blocks required to produce an encoded block reduces the encoding operations, and, also, results in sparse matrices that can be decoded efficiently. The disadvantage of our approach is that some segments may become unpopular. This problem is similar to the optimal scheduling of the block propagation. In the following, we argue experimentally that segment scheduling is a much easier problem than block scheduling and that the penalty of group coding is small.

To study the effect of group coding, we define server efficiency as the number of useful (i.e. unique) blocks that are served by the server in time sufficient to serve one full copy of the file (under optimal scheduling). We measure server efficiency as a ratio and notice that the efficiency of network coding is 1.0; similarly, the efficiency of an ideal block propagation scheme is also 1.0. We estimate the efficiency ratio of group network coding and of unencoded transfers using the traces we have collected in our experiments.

We concentrate on the block transfers that took place from the beginning of the download until the server has transmitted a number of blocks equal to the number of blocks in the file (e.g. 1500 blocks in Trial-4). Observe that we also include block transfers between nodes. We assume that the underlying topology and peer matching remains the same as we evaluate different block scheduling techniques. We measure the number of unique blocks transmitted by the server. In the case of group network coding, the first b blocks from a particular segment are unique, where b equals the number of blocks per group; the rest blocks transmitted in that segment are redundant. We have performed 5 runs per instance and we report the average efficiency results in Table 3.

Table 3: Utilization of server resources.

Method	Efficiency
Group coding (50 blocks per group)	0.9552
Group coding (100 blocks per group)	0.9598
No coding, Local rarest, 4 neighbors	0.8863
No coding, Local rarest, 8 neighbors	0.9445
No coding, Random	0.7625
Optimal network coding	1.0000

First it is interesting to node that the efficiency of the system did not suffer when restricting coding over a segment of size 50-100 blocks. We used network coding inside a given segment, but scheduling decisions across different segments were made at random. In fact, the efficiency was very close to 1; that of an optimal coding scheme spanning the whole file.

For the case of local rarest with no coding we have varied the size of the neighborhood used to calculate the rarest first block from 4 to 8 nodes. From Table 3, we observe that a random block choice policy performs very poorly; almost 24% of the tranmissions are redudant. We also observe that group coding incurs a penalty of 4.5%, compared to an optimal network coding system, however, it still performs better than no coding. In the case of no coding, improving the nodes' knwoledge about blocks in other parts of the network, improves the efficiency of the server. If the number of neighbors is large enough (8 in our example), then unencoded block transfers result in similar efficiency as group coding. This is due to the limited number of participants in our experiments. We expect that in larger networks, the local rarest heuristic will be a poor estimator of the state of the network (in the same spirit that local rarest with 4 neighbors performed worse than local rarest with 8 neighbors).

We conjecture that the number of neighbors that need to be sampled to get a reliable estimate of the state of the network is not constant with the size of the network. Hence, without encoding, we may need a large number of connections to guarantee efficient utilization of the server resources. On the other hand, group network coding results in good performance even if the average node degree is small (in the case of group network coding in our experiments, the peers could not see the state of their neighbors).

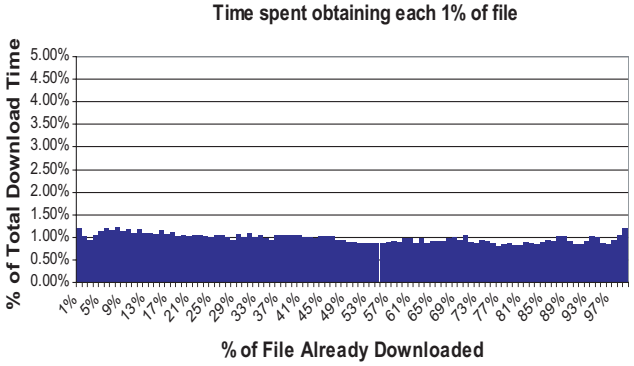

Figure 8: Amount of time spent at each stage of the download.

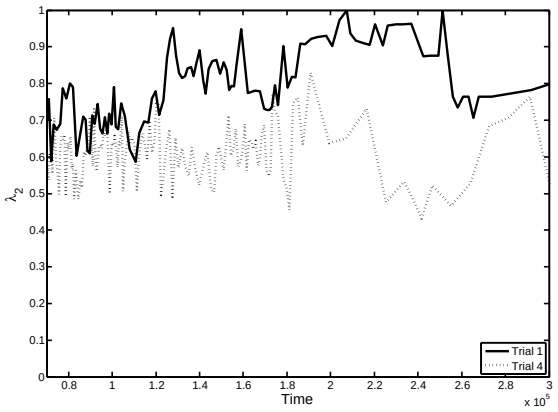

Figure 9: Evolution of the second eigenvalue λ_2 for the (overlay) topologies of Trials 1 and 4.

4.8 Download Progress

Anecdotal evidence suggests that downloaders in current peer-to-peer systems perceive slow performance in the beginning of the download, because they do not have anything to offer (or whatever they have is already popular in the system),[2] and toward the end of the download, because they cannot find the last missing blocks. In fact, [30] shows that if one plots the number of users downloading a given portion of the file (e.g. the first 5%, the next 5%, etc), it follows a U-shape, with users spending a large amount of time to obtain the first and last portions of the file. This problem is more acute when the number of seeds is small, or when the size of the cloud is very large. Network coding can be used to solve this problem.

In Fig.8 we plot the average time spent obtaining each 1% of file for all users in Trial-4. For example, the 50^{th} column is the elapsed time it took to go from 49% of the file downloaded to 50% of the file. The height of the column shows how much of the overall download time was spent getting that each one percent. Observe the absence of a U-shape in the graph by using Network Coding. The reason is that each encoded block is *unique* and useful to *any* node. Thus, newly arriving nodes can easily obtain useful blocks that in turn they can exchange with other nodes, and nodes at the end of the download do not need to wait long periods before finding their missing blocks.

5. TOPOLOGY CONSTRUCTION

Topology construction and maintenance are critical components of any mesh network. Topologies with poor connectivity and connectivity algorithms that bias the connections result in low network efficiency and may raise some fairness issues (if only some nodes monopolize the resources of the fast peers and/or the server).

During our four trials, we have experimented with a variety of topology construction algorithms. The first two trials used a simple random construction algorithm, where each new node picks a random subset of nodes out of those in the system (registered at the tracker) and connect to them. Upload and download connections were treated separately, and the set of uploading nodes could be different than the set of downloading nodes. Each node attempts to open new connections until their number reaches a certain target (6 for upload, and 8 for download). The maximum number of allowed download connections is higher than the maximum for upload connections to account for the bandwidth asymmetry. Compared to current p2p systems, our system treats upload and download connections separately.

A standard metric for measuring the global connectivity properties of a graph is the second eigenvalue λ_2 of the stochastic normalization of the adjacency metric of the graph [7, 16]. Unlike other connectivity metrics, such as the clustering coefficient, λ_2 measures global connectivity properties and has been associated with various good properties of the graph (such as conductance and scaling [13], efficient sampling [14], and others). Topologies with good connectivity have second eigenvalue bounded away from 1.

In Figure 9, we plot the changes in λ_2 for Trials 1&4. Note that values above 0.85 indicate graphs with poor connectivity characteristics.

From this figure we can see that Trial 1 had consistently bad connectivity properties due to the peer matching algorithm used. We observe that this algorithm creates clusters of nodes, with few inter-cluster connections, and, hence, graphs with poor connectivity. A closer examination of the data also revealed bad fairness properties since fast links were monopolized by a few peers in the system.

In Trials 3&4, the topology construction algorithm periodically dropped one of its peers at random. We experimented with various values of the frequency of dropping connections and with various policies for choosing the connections to be dropped. We will now show the results for the case where we drop one peer after five blocks are uploaded (assuming that the peer-set was at its maximum). We also tested preferentially dropping slow peers, and got similar results. Our motivation is to force rewiring of connections and, hence, construct overlay topologies with better properties [14, 15, 25, 31]. However, even doing so, we still observed cluster formation. This can be seen in table 4 where we show the breakdown of the connection statistics for Trial 3 and Trial 4. Observe, that during Trial 3 there are a large number of connections established that do not result in any block transfer since they are immediately terminated with a 'too busy' message. The reason being that peers that were just dropped in one part of the network try to contact other peers in other parts of the network, however, they get continuously rejected since other nodes have their connection budget used and are not accepting more connections.

[2]Recall that many P2P systems implement tit-for-tat algorithms to discourage free-riders.

	Trial 3	Trial 4
Total Connect Attempts	88,179	42,218
Failed (e.g. NATs)	72,371	35,233
Established	32,033	14,960
Downloaded Block	12,687	12,131
Too Busy (cluster)	14,905	0

Table 4: Connection Statistics

To avoid this problem, in the Trial 4 we allowed a given peer's *peer-set* to grow temporarily by some threshold (e.g. *max-peer-set* + 2), creating an "elastic" maximum. This enabled dropped peers to get immediately accepted by peers in other pars of the network. The accepting peer would then be in a situation where he would have more peers than the maximum allowed *max-peer-set*, and would try to come back to the max value by randomly dropping other excess peers.

Figure 9 shows the connectivity graph in this scenario. We can see that compared to Trial 1, Trial 4 resulted in very well connected graphs with low λ_2, and this was achieved at the cost of slightly higher churn. Similarly, from Table 4, we can also observe that compared to Trial 3, Trial 4 ensures that peers can find other accepting peers in the network much easier (note the zero 'too busy' messages).

6. CONNECTIVITY

The wide deployment of Network Address Translation (NAT) devices and firewalls reduces peer-to-peer network performance. Peers behind NATs and firewalls, which we shall collectively call *unreachable* peers, cannot receive inbound connections. (We exclude from our definition peers behind NATs and firewalls configured to allow incoming connections.) Unreachable peers cannot exchange content with each other, and, hence, cannot take advantage of the network capacity that exists between them.[3] Both their download performance and the overall system throughput is reduced as a result.

Based on the observed peer performance and the percentage of unreachable nodes, we calculate a) the optimal throughput of the system assuming all nodes are reachable, and b) the optimal throughput of the system taking into consideration the set of unreachable peers. The optimal throughput at time t is computed as the sum of the peak upload rate of all active peers at time t. To compute the system throughput taking into consideration unreachable nodes, we replayed the traces collected during the trials, calculating the optimal throughput given the existing connectivity constraints. To this extend, for each time t we first saturate the upload (or, download capacities) of the plausible connections between unreachable nodes and reachable nodes. Then, we saturate the remaining upload/download capacities of reachable nodes by matching them with each other. This matching is optimal. Our computation of the optimal throughput does not assume an upper limit on the number of connections per node, which can overestimate the computed optimal throughput.

In Fig. 10 we plot the optimal throughput with full node connectivity and with the actual connectivity seen during two different trials. During the first trial considered (Fig. 10(left)), the average number of unreachable peers was quite high, more than 75%; the second trial had less than 60% unreachable peers, possible due to our efforts to educate the users of the performance benefits of configuring their NAT boxes and firewalls.

Observe the large discrepancy between the maximum system throughput with and without considering the unreachable peers in the first trial around time 30hr. After examining the connectivity pattern of users, we realized that at this specific time, the system reached high percentages of unreachable peers (more than 85-90%).

In Fig. 10(right), we present the results for Trial 4 (with less than 60% unreachable nodes). We observe that the throughput under partial connectivity is fairly close to that achieved with full connectivity, which implies that the system performs surprisingly well even with a large number of unreachable peers. We attribute the resilience of the network to two factors: a) the aggregate upload capacity of the high-capacitated, globally-reachable nodes can be saturated by the aggregate download capacity of the unreachable nodes, and b) the aggregate upload capacity of the unreachable nodes can be saturated by the very high download capacity of the few well-connected and fast nodes.[4] We have validated both assumptions analytically and experimentally, but, due to space constraints, we omit the details.

7. SECURITY

A common concern about network coding is the protection against malicious users. Unlike unencoded transmission, where the server knows and can digitally sign each block, in network coding each intermediate node produces "new" blocks. Homomorphic hash functions can be used to verify the encoded blocks using the hashes of the original blocks, however, such functions are computationally expensive [24]. Our scheme is based on the use of random masks and mask-based hashes, which we refer to as Secure Random Checksums (SRCs). SRCs provide signatures capable of verifying the integrity of encoded blocks on-the-fly at a low computation cost. SRCs also have the nice property that they work well with Galois Fields (and not just with modular fields, which are known to be produce more expensive operations, as is the case with homomorphic hash functions).

We now give a high-level explanation of how SRCs work. To produce an SRC, the server creates a vector $r = [r_1 \dots r_m]$ of random elements in Z_q (often $q = 16$ digits). The size of the vector m is the number of *symbols* per block (for a block of 2 MBytes m is $2 * 1024^2$). Then, the server performs pairwise multiplication of the vector of random elements with the vector of symbols of a particular block and adds the results in $GF(2^q)$. For example, assume that the symbols of block i are $b_i = [b_{i,1} \dots b_{i,m}]$ and the random numbers are $r = [r_1 \dots r_m]$, then the SRC of block i is $\sum_{j=1}^{m} r_j b_{i,j}$. The same process is repeated for all n file blocks. Together with the SRCs, the random element vector is transmitted (note that the set of random elements can be replaced with the seed used for the random number generator). Because of the linearity of the computation, it is easy to show that the SRC of an encoded block can be computed from the SRCs of the original blocks. In particular, assume an encoded block $e = \sum_{i=1}^{n} c_i b_i$, corresponding to coefficient vector $\vec{c}$. To verify whether encoded block e is corrupted or not, a node applies the random vector to the e block and checks whether the following equation holds:

[3]NAT traversal techniques can solve the connectivity problem [17]; in this work we are interested in investigating the throughput potential in the absence of such techniques

[4] In our system, (unreachable) nodes periodically attempt to initiate upload connections to other nodes in order to fully utilize their upload capacity.

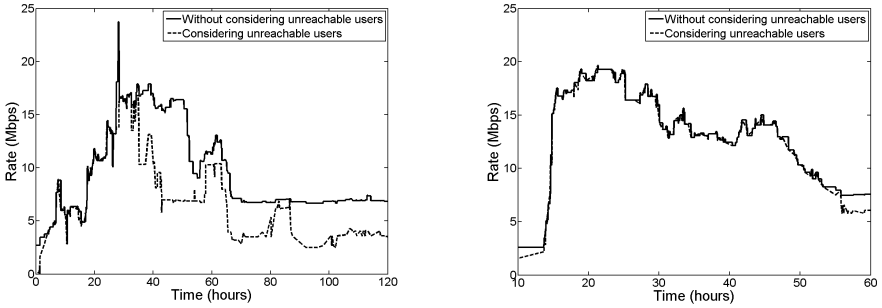

Figure 10: The effect of unreachable peers on the overall performance for two trials (left: Trial-1, right: Trial-4). The top curves are computed assuming all nodes are reachable; the lower curves take into consideration the set of unreachable peers over time.

$$\begin{aligned} \sum_{j=1}^{m} r_j e_j &= \sum_{j=1}^{m} r_j (\sum_{i=1}^{n} c_i b_{i,j}) \\ &= \sum_{i=1}^{n} c_i (\sum_{j=1}^{m} r_j b_{i,j}) \end{aligned}$$

The size of each random element r_j is q bits. Rather than sending the random mask vector, in our implementation, each node downloads the seed used by the server to generate the random mask. Based on the seed, nodes can reproduce the random mask vector $\vec{r}$ locally. The size of the seed is 128 bits, which is negligible. In addition to the seed, each node also downloads the per-block SRC value. The size of the SRCs for all blocks in the file is $n \cdot q$ bits, which results in 2 KBytes for a 1000 block file.

When a new client joins the system, it first contacts the server which computes a new set of SRCs for that client and communicates the SRCs to the client over a secure channel. The client keeps the SRCs secret, since if they are revealed, a malicious node can efficiently fabricate corrupted blocks. A malicious node that does not know the SRCs can trick a node only by pure luck. If the client receives many SRCs,[5] then it is computational infeasible for an attacker to construct corrupted encoded blocks without the corruption being detected.

The SRCs are linear operations and can be computed very efficiently. In our current implementation running on a 3.0 GHz Pentium 4 with 1GB of RAM, SRC generation takes 1 sec/SRC for file of 2 GBytes, which is close to the cost of reading the file once. In the current implementation, the cost of producing SRCs increases linearly with the size of the file. However, more efficient implementations are possible. Note that for a smaller file size (e.g. 200 MBytes), one single server can produce SRCs to serve peers at a rate of 10 peers/sec, or 864,000 users/day. Also, the rate of generation of SRCs at the server is not that critical since it is a process that can happen in the background before the download commences. The rate of SRC verification is close to 1.6 Gbps, which is much faster than the rate at which encoded blocks can be generated. In general, we have observed a negligable impact caused by SRC verification on the nodes performance.

8. RELATED WORK

Understanding and evaluating swarming protocols has been an important topic in the recent years. For instance, [3] compares different swarming strategies and discusses the impact of the number of blocks and the number of simultaneous uploads. They show that the number of chunks should be large and that the number of simultaneous uploads should be between 3 and 5 in order to have good efficiency. Qiu and Srikant [29] provided an analytical solution to a fluid model of BitTorrent with global information. Felber et al. [9] compared different peer and piece selection strategies in static scenarios using simulations. Bharambe et al. [2] presented a simulation-based study of BitTorrent using a discrete-event simulator that supports up to 5000 peers and found different inefficiencies using peer sets lower than 15 peers. Izal et al. [21] provided insights into BitTorrent based on data collected from a popular tracker log from a local peer perspective. Our work differs in that it provides detailed experimental results of a live P2P distribution system from a variety of novel angles by monitoring all its components. In addition, our paper provides details of our experiences with network coding in a P2P environment.

A number of cooperative architectures [23] [4] have proposed the use of Erasure Codes [28] (e.g. Digital Fountain [5]) to efficiently transfer bulk data. However, in such systems the set of symbols acquired by nodes is likely to overlap substantially, and care must be taken to enable nodes to collaborate effectively. This makes cooperation and reconciliation among nodes more difficult than when no content is encoded. 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.

Most of the previous work on network coding is largely based on theoretical calculations that assume a detailed knowledge of the topology, a centralized knowledge point for computing the distribution scheme and focus on multicast environments. However, little effort has been made to build and evaluate the feasibility of network coding on a real setting. In [11] we provided a comparison of different swarming algorithms and evaluated via simulations the performance of a network coding in P2P systems. However, it was believed that network coding could not be made practical in real settings due to its computational complexity and the difficulties protecting against block pollution attacks. Our work focuses on the study of a live network coding P2P system. In a similar spirit, recent work by Katti et al. [22] provides the first implementation results of network coding in Wireless networks. The authors implemented and tested network coding in a mesh wireless network and showed important benefits when multiple unicast flows are mixed.

9. SUMMARY

In this paper we have described our experiences with a P2P system that uses network coding. Based on a prototype implementation of our system and the result of several live distributions, we show that

[5]In our implementation each symbol is 16 bits long, and hence 10 SRCs result in 160 random bits

network coding overhead is relatively small, both in terms of CPU processing and I/O activity. We also describe a scheme for efficient verification of encoded blocks and show that the verification process is very efficient.

Moreover, we measure a high utilization of the system resources and large savings for the content provider even during *flash-crowd* events. We also observed a smooth file download progress (i.e. users do not spend much time in the beginning or the end of the download), and very efficient utilization of the server capacity.

While coding obviates the need for fancy block selection algorithms, the system's efficiency still depends largely on how peers are connected. We provide an initial description of the impact that unreachable nodes can have and show that surprisingly the system is highly resilient to very large number of unreachable peers (e.g. as high as 70%). We also show that the topology construction algorithms can have a significant impact in the overall system performance. However, a deeper analysis is required to better understand the impact of peer-matching algorithms in the system's efficiency (e.g. algorithms that take into account connectivity or access rates to pair nodes).

10. REFERENCES

[1] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 2000.

[2] A. Bharambe, C. Herley, and Venkat Padmanabhan. Analysing and improving bittorrent performance. In *Infocom 2006*, 2006.

[3] Ernst Biersack, Pablo Rodriguez, and Pascal Felber. Performance analysis of peer-to-peer networks for file distribution. In *Fifth International Workshop on Quality of Future Internet Services (QofIS04)*, 2004.

[4] John Byers, Jeffrey Considine, Michael Mitzenmacher, and Stanislav Rost. Informed content delivery across adaptive overlay networks. In *SIGCOMM*, Pittsburgh, PA, 2002. ACM.

[5] John Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data. In *SIGCOMM*, 1998.

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

[7] Fan R. K. Chung. *Spectral graph theory*. Regional conference series in mathematics, no. 92. Published for the Conference Board of the mathematical sciences by the American Mathematical Society, Providence, R.I., 1997. Fan R.K. Chung. 26 cm. "CBMS Conference on Recent Advances in Spectral Graph Theory held at California State University at Fresno, June 6-10, 1994"–T.p. verso.

[8] Bram Cohen. <`http://www.bittorrent.com`>.

[9] P. Felber and Ernst Biersack. Self-scaling networks for content distribution. In *International Workshop on Self Properties in Complex Information Systems*, 2004.

[10] C. Fragouli, J.-Y. Le Boudec, and J. Widmer. Network coding: An instant primer. Technical Report TR-2005-010, EPFL, 2005.

[11] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *IEEE Incofom*, Miami, FL, 2005.

[12] C. Gkantsidis and P. Rodriguez. Cooperative security for network coding file distribution. In *IEEE Infocom*, 2006.

[13] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Conductance and congestion in power law graphs. In *ACM SIGMETRICS*, pages 148–159, San Diego, CA, US, 2003. ACM Press.

[14] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks. In *IEEE Infocom*, Hong Kong, 2004.

[15] Christos Gkantsidis, Milena Mihail, and Ellen Zegura. The markov chain simulation method for generating connected power law random graphs. In *SIAM Alenex*, Baltimore, MD, 2003.

[16] Christos Gkantsidis, Milena Mihail, and Ellen Zegura. Spectral analysis of Internet topologies. In *IEEE Infocom*, San Francisco, CA, US, 2003.

[17] S. Guha and P. Francis. Characterization and measurement of TCP traversal through NATs and firewalls. In *ACM IMC*, 2005.

[18] Tracey Ho, Ralf Koetter, Muriel Medard, David R. Karger, and Michelle Effros. The benefits of coding over routing in a randomized setting. In *IEEE International Symposium on Information Theory (ISIT)*, page 442, Yokohama, Japan, 2003.

[19] BBC iMP. <`http://www.bbc.co.uk/imp`>.

[20] AOL In2TV. <`http://television.aol.com/in2tv`>.

[21] 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. In *Passive and Active Measurements (PAM)*, 2004.

[22] S. Katti, H. Rahul, W. Hu, D. Katabi, J. Crowcroft, and M. Medard. Network coding made practical. <`http://nms.lcs.mit.edu/~dina/pub/cope-3.ps`>, 2006.

[23] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Symposium on Operating Systems Principles (SOSP)*, 2003.

[24] M. N. Krohn, M. J. Freedman, and D. Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *IEEE Symposium on Security and Privacy*, 2004.

[25] Ching Law and Kai-Yeung Siu. Distributed construction of random expander networks. In *IEEE Infocom*, San Francisco, CA, USA, 2003.

[26] Zongpeng Li, Baochun Li, and Lap Chi Lau. On achieving maximum information flow rates in undirected networks. *Joint Special Issue on Networking and Information Theory, IEEE Transactions on Information Theory and IEEE/ACM Transactions on Networking*, June 2006.

[27] David Mackay. Information theory, inference, and learning algorithms. In *Cambridge University Press*, 2003.

[28] Petar Maymounkov and David Mazires. Rateless codes and big downloads. In *IPTPS'03*, 2003.

[29] D. Qiu and R. Srikant. Modeling and performance analysis of bittorrentlike peer-to-peer networks. In *ACM SIGCOMM04, Portland*, 2004.

[30] Ye Tian, D. Wu, and K.-W. Ng. Modeling, analysis and improvement for bittorrent-like file sharing networks. In *IEEE Infocom*, 2006.

[31] V. Vishnumurthy and P. Francis. On heterogeneous overlay construction and random node selection in unstructured p2p networks. In *IEEE Infocom*, 2006.