Anatomy of a P2P Content Distribution system with Network Coding

Christos Gkantsidis, John Miller, Pablo Rodriguez
Microsoft Research, Cambridge
email:{chrisgk, johnmil,pablo}@microsoft.com

ABSTRACT

In this paper we present our experiences with a P2P content distribution system that uses Network Coding. Using results from live trials, we are able to present a detailed performance analysis of such P2P system from a variety of novel perspectives. We show that Network Coding incurs little overhead, both in terms of CPU processing and I/O activity, and it results in smooth and fast downloads. To ensure secure transfers, we describe a novel scheme that verifies encoded blocks on-the-fly and analyze its performance. We also study the effect of peers behind NATs and firewalls and show the robustness of the system to large number of unreachable peers.

1. INTRODUCTION

In recent years, a new trend has emerged with peer-to-peer (P2P) systems providing a scalable solution for distributing commercial, legal content (e.g. [4, 8]). 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 [4], 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. In this paper, we present our experiences implementing and using our system. In particular:

a) We present the performance of a live P2P filecasting system from a novel set of angles (§3). Our system collects statistics from both the server and the peers, and, as a result, allows us to 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 (§3.5). Moreover, coding is effective at eliminating the first/last-blocks problems (§3.6).

c) We study the influence of unreachable nodes (e.g. behind NATs, firewalls) in the system's efficiency (§4). We compare the percentage of unreachable nodes with the system's efficiency over time, and observe that the system is highly resilient to large number of unreachable peers (e.g. as high as 70%).

d) We study a novel set of security functions to secure Network Coding systems, which we call *Secure Random Checksums* [5], and show that they have a negligible computational overhead and, hence, allow on-the-fly verification (§5).

2. SYSTEM OVERVIEW

2.1 Network Coding

Network coding is a novel mechanism that promises optimal utilization of the resources of a network topology [1, 2, 7, 10]. 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 packet can be further combined to generate new linear combination. The original information can be reconstructed after receiving enough linearly independent packets.

This is of great use in large-scale distributed systems, such as P2P networks, where finding the proper scheduling of information across the overlay topology is very difficult. Compared to traditional approaches, network coding makes optimal use of the available network resources 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 [4]. An overview of network coding and its applications is given in [3].

2.2 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 [2], 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. RESULTS

3.1 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

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.

3.2 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. 3. 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

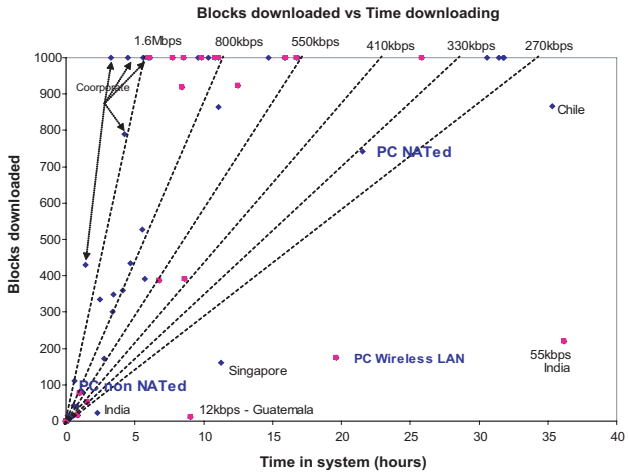

Figure 1: Summary of participating users (Trial-1).

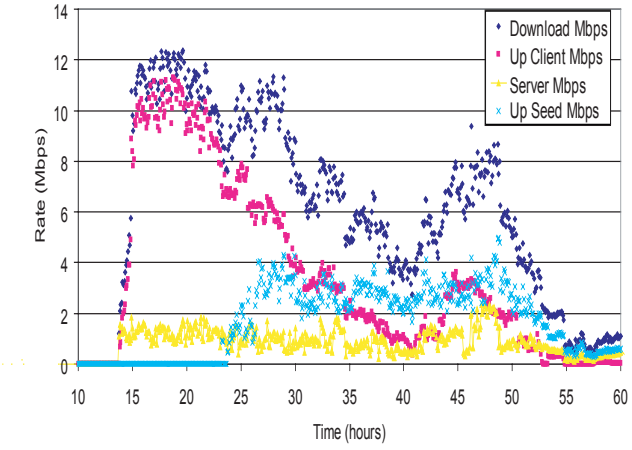

Figure 3: System Rates over time.

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 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.

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

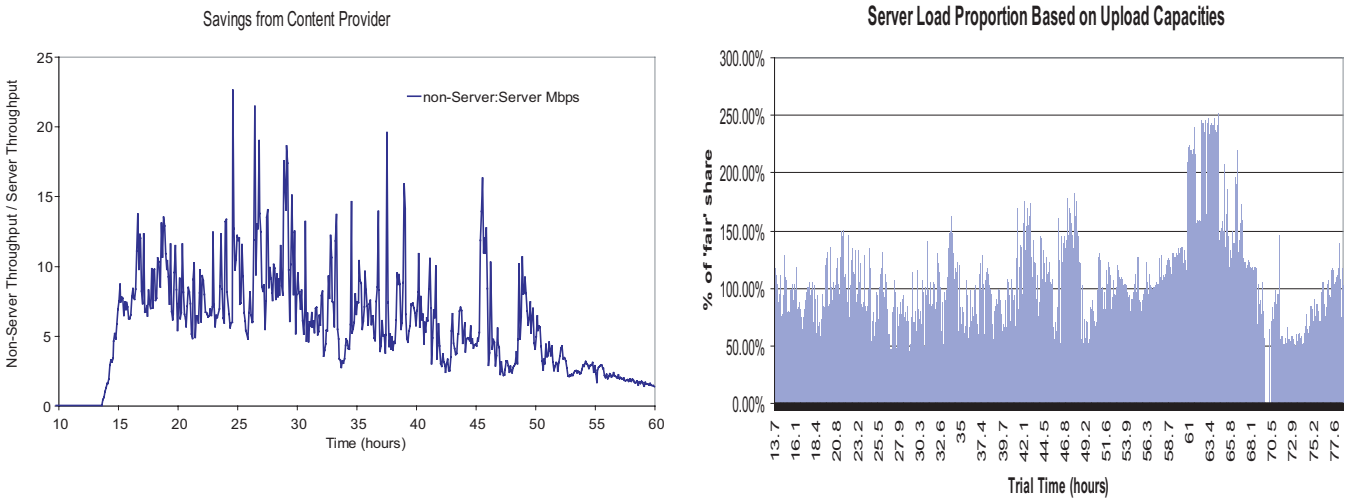

Figure 2: Content Provider effort. Left: Content Provider Savings. Right: Server Share.

3.3 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. 2(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. 2(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.

3.4 Peer's Performance

We now focus on the performance seen by a typical peer. In Fig. 4(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. 4(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 the 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.

3.5 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. 5(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 deep 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. 5(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 consumption are minimal. We expect the overheads to become negligible as we implement more sophisticated encoding and decoding techniques.

3.6 Download Progress

Anecdotal evidence suggests that downloaders in current peer-to-peer systems perceive slow performance in the be-

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

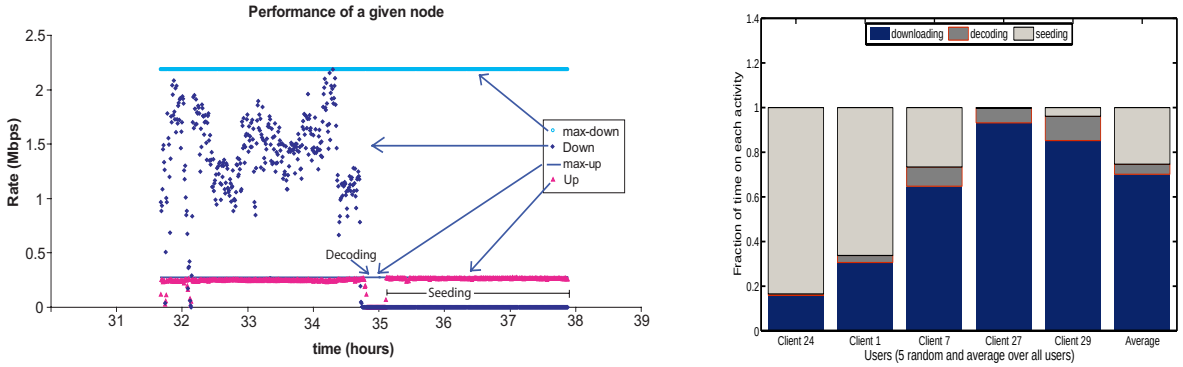

Figure 4: Description of a peer's activity and performance. Left: Peer download and upload rates. Right: Time on each activity (for 5 random peers and average).

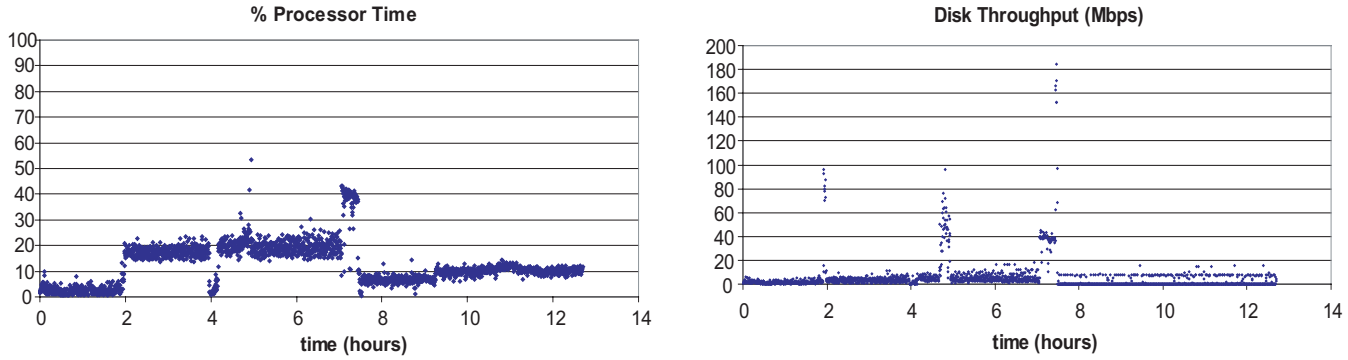

Figure 5: Resource consumption on a typical machine during Trial-4. Left: CPU Activity. Right: Disk Activity.

ginning of the download, because they do not have anything to offer and get low priority,[2] and toward the end of the download, since they cannot find the last missing blocks. In fact, [11] 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.6 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 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.

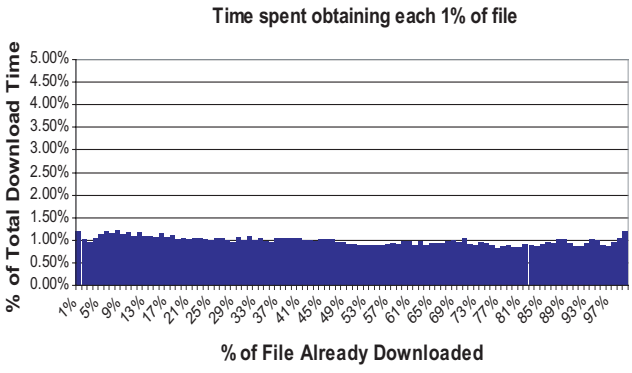

Figure 6: Amount of time spent at each stage of the download.

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

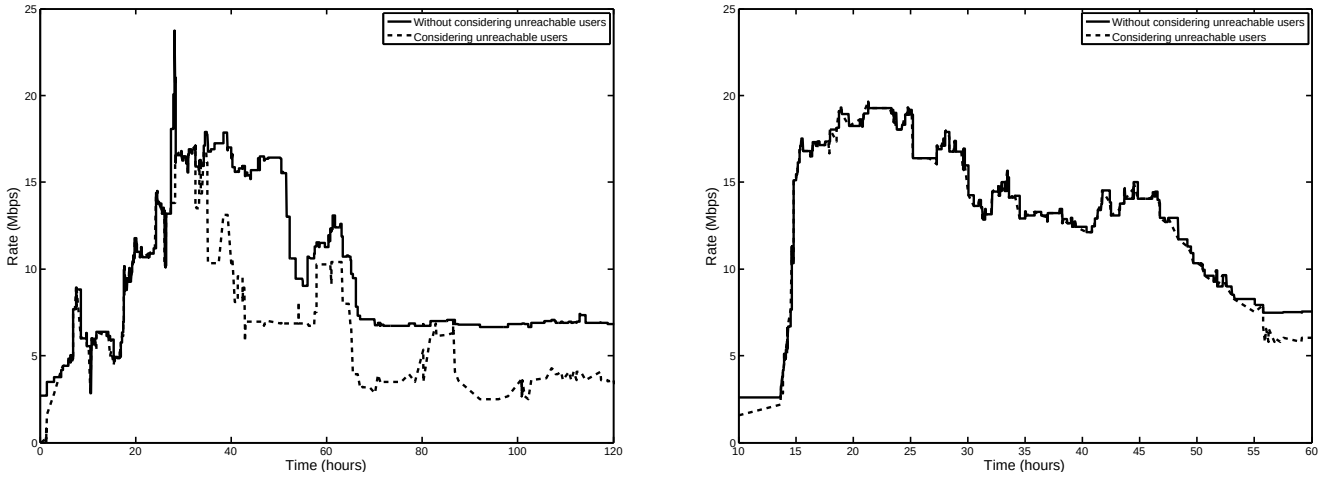

Figure 7: The effect of unreachable peers on the optimal aggregate throughput of the system for two trials. Left: Trial-1, Right: Trial-4. The top curves are computed assuming all nodes are reachable; the lower curves take into consideration the set of unreachable peers over time. In Trial-1 the percentage of unreachable users much higher compared to Trial-4.

4. 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 extent, 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. 7 we plot the optimal throughput with full node connectivity and with the actual connectivity seen during two different trials. In Fig. 7(left), we present the results for Trial 1, where the average number of unreachable peers was quite high, more than 75%. 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%), which are responsible for the low throughputs.

In Fig. 7(right), we present the results for Trial 4, which had less than 60% unreachable peers, possibly due to our efforts to educate the users of the performance benefits of configuring their NAT boxes and firewalls. We observe that the throughput under such 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) a few nodes with high capacities and good connectivity can provide good throughput for most unreachable nodes, and b) typically the upload access capacity of a node is much smaller than the download, hence, even-though unreachable nodes can accept incoming connections, they can connect to other reachable nodes and saturate their download capacities. We have validated both assumptions analytically and experimentally, but, due to space constraints, we omit the details.

5. 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 [9]. In [5], we propose another verification process that requires extra work from the server, but is extremely efficient to compute, communicate, and check. Our scheme is based on the use of random masks and mask-based hashes, which we refer to as Secure Random Checksums (SRCs). Moreover, our proposed scheme can work with efficient Galois Fields (and not only with more expensive Z_p fields as is the case with

[3]Note that recent NAT traversal techniques provide efficient solutions to alleviate the connectivity problem [6]

homomorphic hash functions).

We now give a high-level explanation of how SRCs work. The server produces as many random elements (numbers in the encoding field) as the number of symbol elements in each block. Then, it performs pairwise multiplication of the vector of random elements with the vector of block elements and adds the results. For example, assume that the symbol elements of the block are $B_i = [b_{i,1} \dots b_{i,n}]$ and the random numbers are $r = [r_1 \dots r_n]$, then the SRC of block i is $\sum_{j=1}^{n} b_{i,j} r_j$. The same process is repeated for all file blocks. An SRC description is the set of the random elements plus the SRCs of each of the blocks of the file (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.

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,[4] 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. Our prototype checks encoded blocks at a rate of almost 100Mbps on a Pentium 4 at 3GHz with 1GB of RAM. The server can generate new SRCs at a rate higher than 20 Mbps. Note that 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.

6. 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 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 (proposed originally in [5]), 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.

While coding obviates the need for fancy block scheduling 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%). 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).

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

REFERENCES

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

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

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

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

[5] C. Gkantsidis and P. Rodriguez. Cooperative security for network coding file distribution. In *IEEE Infocom*, 2006 (to appear).

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

[7] 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.

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

[9] 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.

[10] 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.

[11] Ye Tian and K.-W. Ng. Analysis and improvement for file sharing networks. (To appear at Infocom 2006) <`http://appsrv.cse.cuhk.edu.hk/~ytian/pub/BTIncentive.pdf`>, 2005.