Peer-Assisted Content Distribution with Random Linear Network Coding

Hundebøll, Martin; Ledet-Pedersen, Jeppe; Sluyterman, Georg; Madsen, Tatiana Kozlova; Fitzek, Frank

Published in:
Vehicular Technology Conference (VTC Spring), 2014 IEEE 79th

DOI (link to publication from Publisher):
10.1109/VTCSpring.2014.7023041

Publication date:
2014

Document Version
Early version, also known as pre-print

Peer-Assisted Content Distribution With Random Linear Network Coding

Martin Hundebøll, Jeppe Ledet-Pedersen, Georg Sluyterman, Tatiana K. Madsen, Frank H.P. Fitzek
mhu@es.aau.dk, jlp@satlab.org, georg@sman.dk, {tatiana,ff}@es.aau.dk
Department of Electronic Systems, Aalborg University, Denmark

***Abstract*—Peer-to-peer networks constitute a widely used, cost-effective and scalable technology to distribute bandwidth-intensive content. However, the majority of todays peer-to-peer systems require complex algorithms to schedule what parts of obtained content to forward to other peers. Network Coding (NC) has been proposed as a method for optimizing this scheduling and previous work supports that a gain in overall throughput may be possible. Networks that apply NC differs from store-and-forward networks by distributing content as a linear combination of received packets.**

In this paper we propose the structure of the BRONCO peer-to-peer system, which applies random linear network coding. To support simulation results in related studies, we focus on experimental evaluation of performance. Our protocol is implemented using coding in the binary Galois field, which is computationally efficient compared to coding in higher order fields. The overlay network is easy to establish and nodes communicate using a simple protocol.

We show that BRONCO outperforms regular HTTP transfers and is, even with a simple protocol, comparable with BitTorrent. Furthermore, we evaluate the performance of different coding parameters and suggest a suitable trade-off between CPU utilization and network overhead. Within the limitations of the used test environment, we have shown that NC is usable in peer-assisted content distribution and we suggest further improvements to reduce redundancy overhead.

***Index Terms*—Content Distribution, Implementation, Network Coding, Overlay Network, Peer-to-peer**

I. Introduction

When distributing content from one source to multiple destination nodes, simple file transfer protocols such as FTP and HTTP are limited by the upload capacity of the source. A method to obtain more efficient transfers is peer-to-peer networks, where the destination nodes cooperate on the distribution of information. Peer-to-peer networks have proved to be a cost-effective and widely used method for distribution of bandwidth-intensive content and are especially tolerant to rapid increases in the number of connected clients, generally known as flash crowds[1]. Examples of peer-assisted services include software updates for the World of Warcraft multiplayer game[2], distribution of the Ubuntu Linux operating system[3], and Skype Internet Telephony[4].

As illustrated in Figure 1, peer-to-peer networks differs from conventional content distribution by forming a logical, cooperative overlay network, where data is shared between nodes and thus taking advantage of the resources available on the destination nodes. The formed network is scalable in the sense that when the capacity demand is increased due to new nodes entering the network, the nodes also provide additional capacity to the network. Nodes may choose to stay in the network even after receiving the complete content, resulting in a further increase of the network capacity.

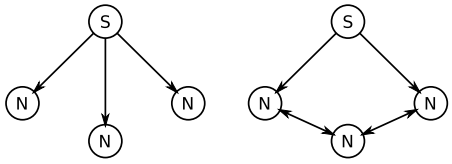

Fig. 1. Left: Content distribution using conventional client-server systems. Right: Content distribution using peer-to-peer distribution.

The majority of todays peer-to-peer networks for file distribution (e.g. BitTorrent[5] and Gnutella[6]) operates by dividing the content in blocks, which are then stored and forwarded by nodes in the network. This allows nodes in the network to concurrently download multiple blocks from different nodes and saves bandwidth costs on the source, as this only needs to deliver content into a subset of the network. Traditionally, the scheduling of the blocks a node transmits to other nodes have required complex algorithms and knowledge of the block distribution of connected nodes. One strategy is the *rarest-first* where nodes prioritize the distribution of blocks that are rare among the connected nodes. As this selection is based solely on knowledge of a local part of the network topology, this can lead to suboptimal content distribution because the perception of rare blocks can be very different for connected nodes.

In [7] it is suggested that incorporating Random Linear Network Coding (RLNC) methods in a peer-to-peer content distribution system could lead to a potential gain in the overall network throughput due to simplification of the block scheduling. In networks where RLNC is employed, nodes exchange random linear combinations of received data, contrary to conventional networks where nodes simply forward received content. When a sufficient number

of linear independent combinations have been received, the original data can be reconstructed. This may lead to a theoretical gain in overall network throughput, but at the cost of increased usage of computational resources due to the coding. Furthermore, the randomness of the encoded blocks can result in nodes receiving linear dependent blocks containing no new information. The coding operations are performed in a finite field. In [8] it is shown that using the binary Galois field, GF(2), facilitates very efficient implementations of both encoding and decoding, by accepting a higher probability of generating redundant linear combinations.

Avalanche is a peer-to-peer system with RLNC developed by Microsoft Research. The system is simulated in [7] and a prototype implementation in C# is evaluated in [9]. Avalanche implements network coding in GF(2^{16}), which results in approximately 20% CPU utilization while downloading, 40% utilization while decoding, and 10% utilization while seeding (Pentium IV 2 GHz, 512 MiB RAM). The performance validation only mentions a download duration of 5.2 hr but nothing on the specific file size. From the papers test specification it is specified to be between 2.8 GiB to 3.7 GiB, yielding a download speed of 157 kB/s to 207 kB/s.

In this paper, we focus on an experimental evaluation of the feasibility of using NC as an approach for content distribution. We propose the structure of a peer-to-peer file distribution network with RLNC in the binary Galois field. By employing RLNC methods, the network allows for easy block scheduling with a very simple protocol.

We have structured the rest of the paper as follows. Section II outlines the scenario and presents the BRONCO (BRONCO Random Overlay with Network Coding Optimization) protocol, explaining how peers connect, exchange data and leaves the network. A prototype implementation in C++ is described and compared with existing protocols in Section III, which also contain tests to support the selection of NC parameters. Conclusions and suggested subjects for further work is given in Section IV.

II. BRONCO: Random Overlay with Network Coding Optimization

In this paper we consider the scenario, that one server with limited upload capacity distributes one file to multiple nodes with a total download capacity greater than that of the server. This scenario is a clear example of a situation where peer-to-peer technology can provide a gain in throughput. The nodes downloading the file should take part of the content delivery network themselves. This is conceptually illustrated in Figure 2. The objective is to put the load in the network of nodes having requested the file, lowering traffic from the server and giving an overall performance gain.

A. Network Topology

To facilitate a simple protocol and implementation, the overlay network is designed to be easy to establish and computationally inexpensive to maintain. The network is defined around a centralized server and multiple nodes arranged in a randomly formed, partially connected mesh network. The server holds the file for distribution and maintains a list of all active nodes in the network. Figure 2 illustrates how nodes are randomly connected to both the server and other nodes. The random topology is simple to construct and can easily cope with peers leaving the network as remaining peers can simply request additional peers from the server.

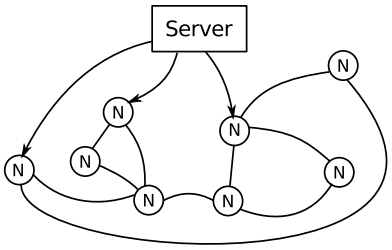

Fig. 2. The topology of the BRONCO overlay network.

Nodes communicating with the server are denominated *clients* and nodes communicating with other nodes are denominated *peers*. During the file transfer the server acts exactly as a peer and follows the data exchange protocol outlined in Section II-C. All peers, including the server, are identified by a 40-byte SHA1 hash generated by the individual peer. The hash should be generated from data unique to the node, e.g. IP address, listen port and startup time. The original file is also identified by a 40-byte SHA1 hash of the content.

BRONCO uses two distinct application layer protocols. The Overlay Network Protocol is used for communication between clients and the server, while the Data Exchange Protocol is used for inter-peer communication and to control the flow of encoded data. Both protocols use the connection-oriented TCP/IP stack.

B. Overlay Network Protocol

The dedicated Overlay Network Protocol describes how peers join and leave the network by connecting to the server.

1) Joining: To join the network, a connection to the server is established from the client and a `PEER` message is sent to the server. The packet contains the file hash, the peer hash, a flag indicating if the client has the complete file, and the port number other peers should use for establishing connections to the peer. Furthermore, the packet includes the number of existing peers requested by the joining peer.

The server replies with a `CONFIG` message containing information of the shared file, as well as packet size and generation size. The server selects the requested number of peers as a random subset of the nodes that are already in the network and returns these to the joining node as a `PEERS` packet. Finally, the joining peer is added to the

list of nodes. The joining node connects to a predefined number of nodes in the received list and proceeds with data exchange as described later in this section. The server itself may be included in the returned peer list with the same probability as any regular peer.

A number of the returned peers may have reached their ingoing connection limit or have left the network without notifying the server, and peers should therefore request more peers than their maximum number of outgoing connections.

2) Leaving: To keep the list of peers in the network up to date, the server is notified when peers leave the network. When leaving gracefully, a peer notifies connected peers and the server with a `LEAVE` message containing its peer hash. The server then removes the peer from the list of peers to prevent it from being returned as part of a peer join.

Due to failure or broken network connection, the peer may also leave the network non-gracefully, in which case the server is unaware of the unavailable peer. Affected peers notifies the server on behalf of the absent peer by sending a `REPORT` message with the peer hash of the missing peer.

C. Data Exchange Protocol

To initiate the content transfer between peers, a joining peer connects to peers in the received peer list. Upon a succesful connect, a handshake is carried out to exchange peer informations. The handshake consists of a `PEER` message from the connecting peer and a `REPLY` message from the remote peer. The remote peer sets a busy flag in the reply, to inform the connecting peer whether the connection is accepted. In addition, the handshake contains peer hashes, file hashes, and a flag indicating if the peer has the complete file. If both peers has the complete file, i.e. when a peer join the network to aid the content distribution only, the connection is discontinued.

If a joining peer is unable to create a connection to a remote peer, this is reported to the server in a `REPORT` message similar to a non-graceful leave.

When the connection is established, both peers may request data by sending a `START` message containing its peer hash. When receiving a `START`, data transfer is started and continues until the receiving peer either sends a `STOP` or `LEAVE` message. Linear combinations are transmitted using `DATA` messages holding the encoded data, the encoding vector and the generation from which the packet is generated.

Peers in the network are in one of three states: *encoding*, *decoding*, or *recoding*. Encoding peers has the complete file and is thus able to create linear combinations from all generations. Encoding peers select the first generation randomly and the following in a round-robin manner to scatter data from all generations in the network. Decoding peers has no connections to incomplete peers and are only processing received data using coding operations. Peers in the recoding state are both receiving and transmitting linear combinations, thus having only part of the file available. When recoding, the peer should generate linear combinations from available generations selected in a round-robin manner. When the file is complete, the peer must inform all connected peers with a `STOP` message.

D. Network Coding Parameters

The original content is organized in generations of g packets, where each packet has size b. The field size used for network coding parameters is denoted q.

Selection of the parameter values g, b, and q poses a trade-off between the computational complexity of the coding and the possibility of generating linear dependent packets. If the size of the original content is kept constant, increasing g or q lowers the expected number of linear dependent encoding vectors, as the total number of valid vectors is also increased. However, the processing required to encode and decode packets increases as well, since more original packets are expected to be included in the encoded packet. The total number of generations also depend on b, the packet size. If the content size requires multiple generations, lowering the packet size will decrease the possibility of receiving packets from a non-complete generation.

E. Implementation

For testing purposes, we have implemented a prototype of the BRONCO peer-to-peer system. The prototype is implemented in the C++ programming language and applies the `libgf2` network coding library provided by the authors of [8]. Each peer connection is running in separate threads, while the network coding is performed by a set of threads to simplify mutual exclusion to the central coding data. Encoded or recoded packets are inserted by the encoding thread in an outgoing buffer. Similarly, connections insert received packets in an incoming buffer to be processed by the decoding thread. Both buffers are organized as fixed-size last-in, first-out queues to give preference to the most recent data.

The defined packets from both protocols are transmitted using TCP/IP connections with data following a predefined header, allowing varying size and type of content. The content field is generated with Google Protocol Buffers[10], which provides a simple, portable, and efficient binary object serialization protocol. The packet header is organized with an 8 bit type field and an 8 bit length field, which permits packets of up to 4 GB and leaves plenty of room to extend the protocol with more packet types.

III. Performance Evaluation

Here we evaluate the performance of the implementation of BRONCO. Before comparing BRONCO to other protocols and evaluating performance with varying parameters, the environment in which the tests are carried out is described. The purpose of the comparison is to state whether BRONCO is a viable protocol.

A. Test Environment

We setup a test environment consisting of multiple controllable and connected nodes and a central server. Furthermore, the bandwidth is limited to each node as well as the server.

The described test environment is obtained with 36 nodes, one server and one router controlling the bandwidth. The server is configured with a 10 Mb/s upload link and each node is configured with symmetric 5 Mb/s links. By selecting the server upload rate smaller than the overall download capacity of the nodes, we are able to test if BRONCO gives a gain in network throughput. The similar link rates makes comparison of results between nodes easier. On the router, IPFW and the Dummynet traffic shaper[11] are used to configure the selected rates and VLANs are set up to control traffic flow.

B. Comparison with Existing Protocols

BRONCO should show an improvement in transfer time over the standard method for file transfers, which we consider to be HTTP downloads. Furthermore, an indication of the performance of BRONCO in comparison to BitTorrent is relevant, since both are based on peer-to-peer technology and BitTorrent is widely used on the internet [12]. We therefore test BRONCO by measuring total transfer times for each protocol.

The test is carried out for HTTP by transferring a 10 MiB file from the server to all nodes, with connections limited as described above. BitTorrent is tested by transferring a 100 MiB file, since slow transfer initialization for BitTorrent gives biased results when comparing with 10 MiB files. The results are summarized in Table I.

	HTTP	BRONCO	BitTorrent	BRONCO
File Size	10 [MiB]	10 [MiB]	100 [MiB]	100 [MiB]
Mean	279.4 [s]	33.8 [s]	292 [s]	291 [s]
Std. Dev.	0.55 [s]	1.7 [s]	2.3 [s]	22 [s]

TABLE I
TRANSFER TIMES FOR HTTP, BITTORRENT AND BRONCO WHEN TRANSFERRING SIMULTANEOUSLY.

As expected, the results in Table I shows that BRONCO clearly outperforms HTTP when multiple nodes download simultaneously. Independent of file size, BitTorrent uses approximately 20 - 30 seconds to establish peer connections, before the transfer is started. This indicates that BitTorrent is better suited for larger transfers, where initialization accounts for less of the total transfer time.

Despite the early prototype of BRONCO, BitTorrent is only marginally faster when initialization is ignored. Taking the maturity of the BitTorrent protocol into account, BRONCO performs relatively well, suggesting that further tests should be carried out to improve BRONCO. The deviation seen when transferring 100 MiB files with BRONCO is caused by the last joining peers struggling to find peers with available connections, thus having lower transfer rate.

C. Scalability

To test how BRONCO scales, transfers with increasing number of nodes are measured and compared to HTTP transfer from a single server. The results in Figure 3 show that the HTTP transfer times increases proportionally to the number of nodes when the download capacity of the clients exceeds that of the server. By utilizing the peers upload capacity, BRONCO shows an approximately constant distribution time

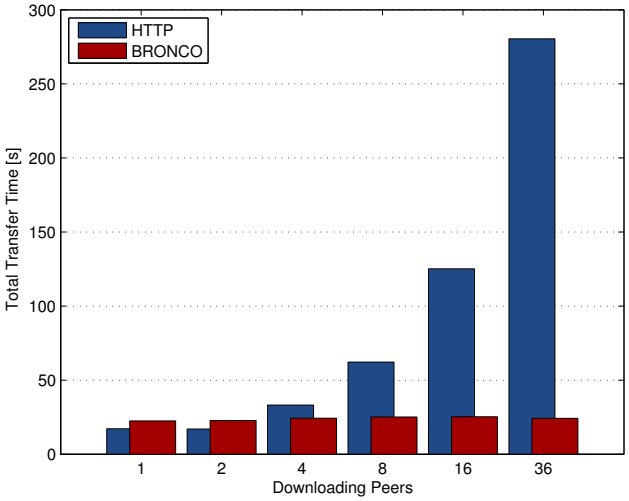

Fig. 3. Comparison of scalability between HTTP and BRONCO when transferring 10 MiB files to increasing number of nodes.

The overhead from redundant packets in the network is apparent when the capacity of the server matches the capacity of the clients, as seen in the tests with one and two nodes.

D. Parameters

We have shown that the BRONCO protocol is scalable and nearly on par with BitTorrent in terms of transfer rate when considering the specific test conditions. Here we evaluate how the generation size affects the performance of the system by addressing the link utilization, CPU utilization, redundant packets share, and the transfer rate. The tests are conducted by distributing a 12.5 MiB file from a server with 10 Mb/s upload capacity. All 36 peers are configured with symmetric 5 Mb/s links and initiate the transfer with a one second interval.

1) Link Utilization: In Figure 4 the average link utilization for all peers is shown for different generation sizes. Total rate is *packets received per second*, regardless of redundancy. Effective rate is received packets providing linear independent packets, which is determined from the change in combined ranks of all generations.

The figure shows that BRONCO is able to receive packets at a total rate close to the full link rate and

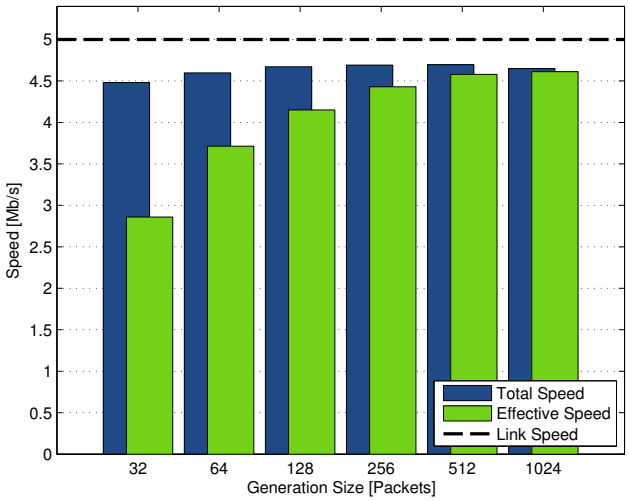

Fig. 4. Link utilization with generation sizes

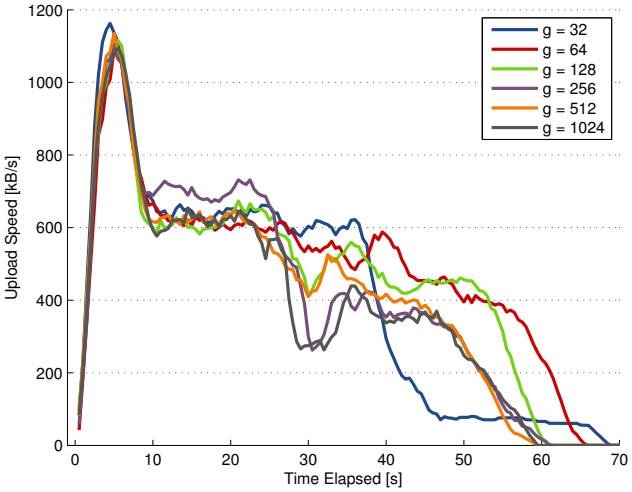

Fig. 5. Server upload rate while peers download.

for higher generation sizes, the rate of redundant packets decreases. This supports that network coding with larger generation size reduces the number of redundant packets, as the probability of generating linear dependent encoding vectors decrease. The residual link capacity is due to slower rates during the beginning of the transfer and framing overhead for TCP/IP and Ethernet.

2) Server Upload Speed: A motivating factor for peer-assisted content distribution is a reduction of data transmitted by the server. In Figure 5 we show the average upload rate of the server. The first peers connecting to the network receive the server as part of the peer list, which is clearly seen by the peak in the region around five seconds. As more peers join the network and data becomes available from peers, the link utilization of the server is decreased.

Generation Size	32	64	128	256	512	1024
Bytes TX [%]	7.4	8.0	8.6	7.7	7.7	7.2

The above table lists the servers share of all data distributed to the peers at different generation sizes. It is seen that with the conditions in our test setup, less than 9% of total data distributed originates from the server. This gives an opportunity for content providers to cut bandwidth costs compared to HTTP. The nearly constant percentage complies with the similar upload rates in Figure 5 and shows that server load is unaffected by change in generation size. CPU utilization of the server is further addressed in Section III-E.

3) Redundant Packet Share: As showed earlier, the probability of receiving redundant packets increases for lower generation sizes. Figure 6 shows the percentage of received packets containing linear dependent encoding vectors, i.e. no new information.

The results in this test are obtained by transferring 12.5 MiB data with a packet size on 6400 bytes, giving equally sized generations for the considered generation

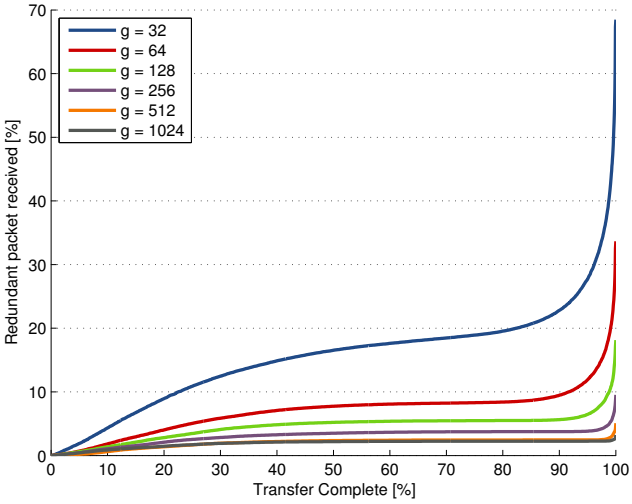

Fig. 6. Redundant packets received during download with different generation sizes.

sizes. Peers transmit packets from a generation selected in a round-robin manner. For configurations where the file size does not divide evenly in bg, the smaller generation is most likely to complete first, rendering the following packets from this generation redundant, thus increasing the total share of redundant packets.

From Figure 6 it is easily seen that the redundancy is drastically increased during the end of the transfer when only a few packets from each generation are missing. To reduce this behaviour, one solution is to introduce an *end-game strategy* which allows peers to request a specific uncoded packet.

4) Client CPU Utilization: As discussed earlier, coding in the binary Galois field can give a performance gain compared to higher field orders. Figure 7 illustrates the average CPU utilization for the clients when de- and recoding data during file transfer.

The decoder uses the Gauss-Jordan algorithm to invert the encoding matrix. The linear trends in the figure are caused by the increasing number of row operations performed for each received packet. The sudden drop near 95% is caused by the increased number of received redundant packets, as discarding a packet when identified as redundant is computationally simpler than full decoding.

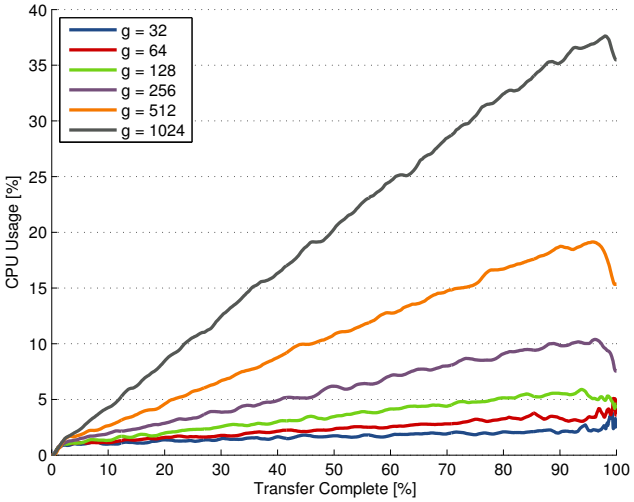

Fig. 7. CPU usage of peers while download at different generation sizes. (3 GHz Pentium 4)

Comparing Figure 6 and Figure 7 shows the trade-off between redundancy and CPU utilization. Accepting a higher CPU utilization, i.e. increasing the generation size, allows lower redundancy.

5) Transfer Progress: By illustrating the average share of time spent receiving one percent of a file, Figure 8 summarizes the performance of BRONCO with a generation size of 256. When joining the network, a short time is spent establishing connections to peers, which is seen as the increased transfer time of the first percentage. From 3% to 95% of the transfer, the figure shows that BRONCO receives data at a steady rate. The need for an end-game strategy is supported by the figure, as the final two percent of the file requires eight percent of the total transfer time.

E. Coding Performance

In this section we evaluate the maximum coding throughput in situations where the network connections between peers is not the limiting factor. Two nodes in the test setup are configured as client and server and a 250 MiB file is transferred from the server with varying generation size. The results of this test is listed in Table II.

As expected, both the average and peak encoding/decoding speed are decreased when the generation size is increased as more matrix operations are required for both encoding and decoding. The clients experience CPU loads of more than 90% for all generation sizes, suggesting that the effective transfer speed is limited by the performance of the decoder. This is further supported

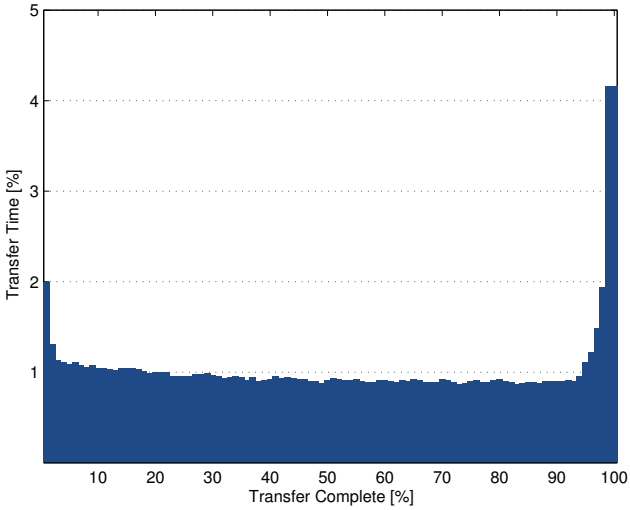

Fig. 8. Percentage of download time used on download one percentage of the file with generation size 256.

Generation Size	32	64	128	256	512	1024
Avg. Speed [MiB/s]	35.0	29.1	18.8	10.4	5.6	3.0
Peak Speed [MiB/s]	53.4	39.9	26.2	15.2	8.9	5.0
Redundant Packets [%]	32.7	14.5	8.7	3.4	1.4	0.7
Client CPU Util. [%]	97.9	98.5	99.5	94.9	94.6	92.5
Server CPU Util. [%]	72.3	75.4	72.3	69.5	65.6	63.6

TABLE II
CODING MEASUREMENTS OF 250 MIB TRANSFER BETWEEN TWO PEERS USING AN UNCAPPED 1 GB/S LINK.

by the server CPU load decreasing for higher generation size, as the client struggles to decode the received data. The share of redundant packets is approximately half of the ones included in Figure 6, due to the peer only receiving packets from one other peer.

These results indicate that the generation size should be selected based on both the processing power of the peers and the link speed. If e.g. the transfer speed between peers is limited by a 100 Mb/s link, lowering the generation size to less than 256 will not increase transfer speed but only the share of redundant packets.

IV. CONCLUSIONS AND FURTHER DIRECTIONS

In this paper, we have proposed the structure of a peer-to-peer content distribution system based on random linear network coding. Our system, BRONCO, uses a simple overlay network topology which facilitates easy scheduling of block propagation. By implementing the system, we have evaluated the performance of BRONCO in a real life network with finite computational resources. The system effectively reduces link usage of the server and decreases overall transfer time by utilizing bandwidth resources available in the peers. This demonstrates that peer-to-peer systems with network coding is possible in practical implementations with acceptable CPU utilization and transfer speed.

By performing the coding operations in the binary Galois field, the computational requirements of the coding are lowered. Compared to Avalanche, which transfers files at approximately 2 Mb/s with a CPU utilization between 20% and 40%, our prototype, when configured for a generation size of 256, can transfer files with 5 Mb/s at a CPU utilization of approximately 5% and a redundant packet overhead of 9%.

Our results are limited by the conditions of the test environment, where nodes are configured with symmetric links. Additional tests should be carried out in setups with more detailed configurations to evaluate BRONCO in an environment with conditions more similar to the internet.

The simple protocol forms a basis for the development of an improved protocol where the rate of redundant packets is reduced. The high rate of redundant packets during the end of the transfer can be reduced by including an end-game strategy, that allows peers to request specific packets to be transmitted. A further reduction can be obtained by avoiding the recoding of packets, when no new content has been received.

In addition, the protocol can be improved by specifying how generation size and packet size should be selected in order avoid the extra redundancy introduced by uneven generations.

References

[1] J. F. Buford, H. Yu, and E. K. Lua, *P2P Networking and Applications*. Morgan Kaufmann Publishers, 2009.

[2] "Blizzard Downloader F.A.Q." Blizzard Entertainment, http://www.worldofwarcraft.com/info/faq/blizzarddownloader.html.

[3] "Alternative Download Options," Canocical Ltd., http://www.ubuntu.com/getubuntu/downloadmirrors#bt.

[4] "P2P Telephony Explained," Skype Limited, http://www.skype.com/help/guides/p2pexplained/.

[5] "What is Bittorrent," Bittorrent, Inc., http://www.bittorrent.com/btusers/what-is-bittorrent.

[6] "What is Gnutella," RFC-Gnutella, http://rfc-gnutella.sourceforge.net/.

[7] C. Gkantsidis, P. Rodriguez *et al.*, "Network coding for large scale content distribution," in *IEEE INFOCOM*, vol. 4. Citeseer, 2005, p. 2235.

[8] J. Heide, M. Pedersen, F. Fitzek, and T. Larsen, "Network Coding for Mobile Devices–Systematic Binary Random Rateless Codes."

[9] C. Gkantsidis, J. Miller, and P. Rodriguez, "Comprehensive view of a live network coding P2P system," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM, 2006, p. 188.

[10] "Protocol Buffers - Google's data interchange format," Google, http://code.google.com/p/protobuf/.

[11] M.Carbone and L.Rizzo, "Dummynet revisited," Università di Pisa, Technical Report, May 2009, available at urlhttp://info.iet.unipi.it/ luigi/papers/20091201-dummynet.pdf.

[12] H. S. . K. Mochalski, "Internet study 2008/2009," Internet, PDF, 2009, http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009.