RapidFlow: An Experimental Testbed for Information Flows with Network Coding

Mea Wang, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
{*mea,bli*}*@eecg.toronto.edu*

***Abstract*— Network coding refers to the capability of coding incoming information flows before transmitting to other nodes in the network, beyond the traditional capabilities of message forwarding and replication on a network node. It has been envisioned that network coding is best applied to overlay networks, in which network nodes are computers at the edge of the Internet and are sufficiently capable and flexible to perform coding operations.**

There exist a wide range of theoretical studies on the benefits of network coding, especially with respect to throughput improvements in the multicast case. However, there has been very little published work on the empirical analysis regarding the benefits and drawbacks of network coding in realistic overlay networks. Towards this objective, we have designed and implemented RapidFlow, a complete and easy-to-use experimental testbed for studying information flows with network coding. In RapidFlow, each node is equipped with a high-performance overlay message switch and a randomized network coding mechanism. In this paper, we present the design and implementation of RapidFlow in an emulated overlay network running on a cluster of workstations, as well as our initial experimental observations with respect to the performance of coded information flows.

I. Introduction

In a *peer-to-peer* content dissemination session, all nodes, except the content source, are interested in receiving the content. To achieve scalable content dissemination, nodes in the network are invited to contribute their upload bandwidth to redistribute content, in an effort to alleviate the load on dedicated content servers. In such a design, such as BitTorrent [1], there does not exist any rigid topological structure among peers. Peers connect with each other at will and are completely flexible to network dynamics due to node failures and joins. However, peers must communicate with each other in order to avoid redundant data being sent to the same node, resulting in excessive communication overhead and lack of flexibility to network dynamics.

In recent work, Gkantsidis *et al.* [2] proposed the idea of facilitating BitTorrent-like peer-to-peer content dissemination using network coding. It was noted that, with a small communication overhead, the application of network coding eliminates the need for reconciliation between peers. Network coding refers to the capability of coding incoming information flows before transmitting to other nodes in the network, beyond the traditional capabilities of message forwarding and replication on a network node. All pieces of information being transmitted are linear combinations of the original pieces. The coding operations are performed in Galois field that preserves the size of the original pieces, *i.e.*, they consume no additional bandwidth, resulting in efficient utilization of network bandwidth. As nodes in peer-to-peer communication sessions are end systems at the edge of the Internet, they are assumed to be computationally capable to perform such coding operations. Therefore, it is feasible to implement such an idea in real-world peer-to-peer communication sessions.

At present, however, the application of network coding in peer-to-peer content dissemination has only been accompanied with simulation-based studies. In this paper, we implement *RapidFlow*, a peer-to-peer content dissemination system with network coding, and use such an implementation to study both advantages and disadvantages of using network coding in peer-to-peer content dissemination. Among other algorithms, *RapidFlow* implements *randomized network coding*, where all coded blocks are linear combinations of the original blocks with random coefficients. It eliminates any additional communication between peers to reconcile their differences.

With the *RapidFlow* implementation, we were able to conduct the first batch of empirical experiments to study the effects of network coding in peer-to-peer content dissemination sessions. We have observed that network coding can conserve up to 20% of bandwidth as compared to a vanilla peer-to-peer protocol in which peers simply act as relays of received content blocks. However, due to the computational overhead of coding, the time to complete a content dissemination session has actually increased, when to compared to our vanilla peer-to-peer content dissemination protocol without using network coding.

The remainder of this paper is organized as follows. Sec. II reviews various architectures of peer-to-peer content dissemination systems and the benefit of network coding. In Sec. III, we present the implementation of RapidFlow, and our initial set of empirical studies using RapidFlow to evaluate the benefits and drawbacks of using network coding in peer-to-peer content disseminations. Sec. IV concludes this paper.

II. Preliminaries

We consider a peer-to-peer content dissemination session, which consists of one source peer and a collection of receiving peers (*receivers*). The objective of the receivers is to receive the content to be disseminated in the shortest period of time, whereas the objective of a source peer is the ability to serve as many receivers as possible. In a conventional client-server

model, both of these objectives are limited by the upload capacity of the source, and the throughput to the receivers degrades as the number of peers in the session increases.

To be more scalable, recent peer-to-peer content dissemination protocols, such as BitTorrent [1], organize nodes into a peer-to-peer network. In these protocols, the source divides the content to be disseminated, *e.g.*, a file in the file system, into n blocks of a fixed size and send these blocks to the receivers. A receiving peer receives these blocks from one or more participating peers, referred to as its *upstream* peers. In return, they also contributes their uplink bandwidth to relay certain received blocks to its *downstream* peers. A receiver eventually reconstructs the original file once it receives all n blocks.

In the ideal case of BitTorrent design, nodes receive disjoint sets of blocks from their peers. In practice, without complete knowledge of the available blocks on peers, a node may receive redundant blocks, leading to a waste of bandwidth. Content reconciliation algorithms have been proposed to improve bandwidth utilization, with the cost of exchanging block availability information between peers, or between a peer and a dedicated "tracking" server.

To further improve the efficiency with respect to bandwidth usage and to minimize the messaging overhead of exchanging block availability information, it has been proposed that participating peers not only relay and replicate received blocks of content, but also code them. Such an idea has been recently proposed by Gkantsidis and Rodriguez in *Avalanche* [2]. It is in line with recent theoretical advances of *network coding* [3], [4], which have been introduced to improve session throughput in directed networks. Based on the principles of network coding, Lun *et al.* [5] has also proposed decentralized optimization algorithms to achieve minimum-cost multicast. These proposals attempt to use network coding to either minimize the cost on the links, or eliminate the needs for content reconciliation between peers, the main source of the communication overhead.

We consider *Avalanche* as an example. Similar to conventional peer-to-peer content dissemination, such as BitTorrent, the file of interest is also divided into n blocks of the same size, referred to as the original blocks $\{b_1, \cdots, b_n\}$. In Avalanche, rather than transmitting or relaying these original blocks, each peer generates and sends coded blocks, which are linear combinations of original blocks. We consider each peer in a peer-to-peer session. The blocks it has received from its upstream peers are $\{b_1, \ldots, b_r\}$, and the blocks it sends to its downstream peers are $\{b'_1, \ldots, b'_t\}$. At the source, $\{b_1, \ldots, b_r\}$ represents the original blocks of the file to be disseminated. A peer uses coding coefficients $\{c_{i,1}, \cdots, c_{i,r}\}$ to produce an outgoing block $b'_i = \sum_{k=1}^{r} c_{i,k} \cdot b_k$, using operations in the Galois field GF(2^k) (usually GF(2^8)). Since all coded blocks $b_i (i = 1, \ldots, r)$ that a peer has received are linear combinations of the original blocks, the newly generated coded blocks b'_i are also linear combinations of the original blocks.

A set of coded blocks are linearly independent if none of them can be expressed as a linear combination of others. Intuitively, any coded block that is linearly dependent of existing coded blocks is considered redundant, since this block can be produced from the existing ones. Therefore, in order to maximize the efficiency of using bandwidth, the set of coefficients must be carefully chosen so that a receiving peer is unlikely to receive linearly dependent blocks. Unless peers cooperate to choose such coding coefficients, it is difficult to design a decentralized and deterministic algorithm to ensure the blocks are all linearly independent with each other. For this reason, *randomized network coding* [6] has been proposed, in which each peer independently generates randomized coding coefficients. *Avalanche* adopts the idea of randomized network coding to produce the coding coefficients.

After a peer has received at least n coded blocks that are linearly independent, it can recover the original blocks by taking the inverse of the coding matrix (which is full rank). The coding matrix is the combination of coefficients used to generate each coded block from a set of original blocks. They are easy to compute in the coding process, and are embedded in each coded block to be sent as one self-contained application-layer message to a downstream peer.

III. Experimental Studies of Network Coding using *RapidFlow*

Though the use of network coding in peer-to-peer content dissemination applications seems promising, to the best of our knowledge, there have not been any experimental studies on the advantages and drawbacks of network coding, using real-world implementations of network coding and data transmissions with actual TCP connections. In this paper, we present our initial experiences and preliminary results with *RapidFlow*, our implementation of a peer-to-peer content dissemination application using network coding. The purpose of implementing *RapidFlow* is to evaluate network coding in realistic peer-to-peer environments.

The RapidFlow testbed is based on a scalable engine of switching application-layer messages from multiple incoming message flows from upstream peers, to multiple outgoing message flows to downstream peers. These message flows can be either stream socket (TCP) or datagram socket (UDP) connections. Our application-layer message switch is designed to consume minimal memory footprint and CPU cycles, and to be flexible to accept a wide variety of message-processing "plug-in" algorithms. The engine is based on our earlier work on *iOverlay* [7], a lightweight middleware framework to facilitate the distributed implementation of overlay algorithms. In addition to the application-layer message switch, we have implemented all the software components required to perform network coding on GF(2^8). All our experiments are conducted on a cluster of dual-CPU servers. We now discuss some additional technical challenges in the implementation of RapidFlow, as well as our initial experiences with RapidFlow.

In the design of RapidFlow, we first need to convey the coding coefficients used in each of the coded blocks to the receiver. As a coded block is a linear combination of the original blocks, it can be uniquely identified by its coding coefficients, referred to as the *signature* of the coded block. A signature can be represented as an array of n coding coefficients $c_k, (k = 1, \ldots, n)$, such that if the original blocks are

$b_k (k = 1, \ldots, n)$, then any coded block $b'_i = \sum_{k=1}^{n} c_k \cdot b_k$. In the implementation of RapidFlow, every time a peer transmits a coded block to its downstream peer, it embeds the signature in the application-level header of a coded block, making it *self-contained*. The overhead introduced by embedding the signature is small, as long as the number of original blocks n is much smaller than the number of bytes s in an original block.

When the size of the content to be disseminated increases, we need to either consequently increase the block size s, or alternatively increase the number of blocks n. Intuitively, if we increase n with the same block size s, the overhead of embedding the signature will be larger. If we increase s with the same number of blocks n, the overhead of embedding the signature will be smaller, but the coding process may be slower. What is the best n and s to be used in RapidFlow in order to achieve the best coding performance?

To answer this question, we performed two simple experiments. We first set $n = 10$ and vary s from 1 KB to 1 MB, and measure the time needed to code. The encoding time is the time taken by a peer to generate a set of randomized coding coefficients and to produce one coded block. The decoding time is the time taken by the receiver to reconstruct the original file from n coded blocks received. Each peer is allocated a dedicated CPU (Intel Pentium IV Xeon 3.6GHz). We observe from the experimental results (Table I) that both encoding and decoding times grow linearly with respect to the block size.

TABLE I

AVERAGE CODING TIMES WITH 10 BLOCKS.

File size (MB)	0.01	0.5	1	2.5	5	7.5	10
Block size (KB)	1	50	100	250	500	750	1000
Encoding time (sec)	0.002	0.051	0.098	0.25	0.491	0.769	0.982
Decoding time (sec)	0.01	0.5	1	2.5	4.9	7.5	10.3

In the second experiment, we set the block size to be 50 KB, and vary the number of blocks from 1 to 10000. The results are shown in Table II. We observe that, when the original file is segmented into more than 100 blocks, the decoding time on a receiver grows rapidly, even with modern processors. Though the growth of encoding times is not as dramatic, it still introduces a considerable amount of delay in the transmission of coded blocks, since they are accumulative from hop to hop. These results have suggested that it may be best for the number of blocks to be very small (around 10) to avoid introducing excessive latencies in peer-to-peer content dissemination sessions. From these two experiments, we conclude that, compared to the block size s, the number of blocks n plays a much more significant role on the encoding and decoding times.

TABLE II

AVERAGE CODING TIMES WITH 50 KB IN EACH BLOCK.

File size (MB)	0.05	0.5	5	50	500
Number of blocks	1	10	100	1000	10000
Encoding time (sec)	0.005	0.05	0.5	10.8	113
Decoding time (sec)	0.005	0.5	59	1330	113000

Next, we consider technical challenges with respect to the generation of coded blocks on each peer. In RapidFlow, the source generates code blocks based on all n original blocks. Each peer generates new coded blocks for their downstream peers as coded blocks are received (and cached locally). A peer may generate coded blocks for each downstream peer using only recently received blocks; it may also choose to code recently received blocks with cached blocks to further reduce the probability of producing linearly dependent blocks. In RapidFlow, we propose to code recently received blocks with a random subset of cached blocks, to maintain reasonable encoding times.

When should a peer start to generate new coded blocks, and how many new blocks should it generate for each downstream peer? Let R_i be the set of coding coefficients of blocks that peer i has received so far. Let $S_{i,j}$ be the set of coding coefficients of linearly independent blocks that i has sent to its downstream peer j. Upon receiving a block, a peer caches it and adds its coding coefficients to R_i if it is linearly independent to the existing ones in R_i. In RapidFlow, a peer i generates one new coded block for each downstream peer j every time it receives a new linearly independent block. In addition, to ensure that all blocks sent from i to j are linearly independent (so that bandwidth is not wasted), i caches coding coefficients of the blocks that it has sent to j in $S_{i,j}$. Peer i then uses its cache as a reference when producing the next coded block for j.

Although each peer never receives linearly dependent blocks from the same upstream peer, it might receive linearly dependent blocks that are sent by different upstream peers. From our experiments, we found that the more upstream peers a peer has, the higher probability a linearly dependent block is received and discarded. Such an observation is due to the shared path from the source to the upstream peers of a particular peer. An example this observation is shown in Fig. 1, where dotted lines represent an overlay path and solid lines represent an overlay link. Since T_2 and T_3 share the same upstream peer T_1, the two coded blocks $c^1_{1,1} \cdot b'_1$ and $c^2_{1,1} \cdot b'_1$ produced by T_1 for T_2 and T_3, respectively, are linearly dependent. Consequently, the coded blocks delivered to T_4 are different linear combinations of the same block b'_1, *i.e.*, they are linearly dependent. The same logic applies to the second batch of blocks arriving at T_4. Therefore, RapidFlow is more bandwidth efficient with a small number of incoming connections on each peer, since upstream peers of a peer are less likely to share a path from the source.

This problem is coupled with our particular implementation of randomized network coding. It can be eliminated by postponing encoding of new blocks until all n blocks are received, which significantly prolongs the downloading time, especially for peers farther away from the source. For this reason, we trade the bandwidth for better download speed since network coding already introduces a considerable amount of delay.

For the purpose of comparisons, we also implement a simple BitTorrent-like file dissemination algorithm, in which a peer randomly forwards received blocks to its downstream peers. For a fair comparison, this algorithm is tested on the same topologies used by RapidFlow.

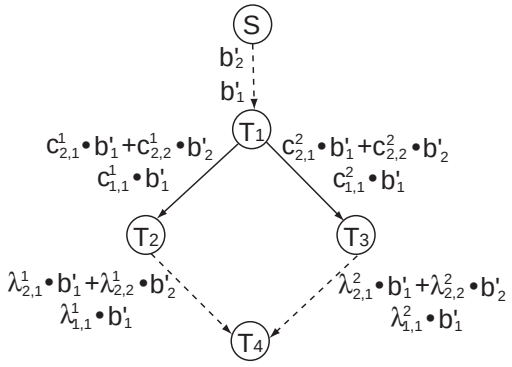

Fig. 1. An example in which linearly dependent blocks arrive at a peer.

We now present an initial set of experiments using RapidFlow, showing the advantage and disadvantage of network coding, in comparison with dissemination without using network coding. Our focus is on the performance of network coding in terms of bandwidth efficiency and the time to complete the downloading process. The experiments are conducted in random topologies, scaled from 10 peers to 100 peers. In order to ensure that every peer can receive the file to be disseminated, we let each peer randomly choose peers that have not received any data as downstream peers. In our typical test runs, the file to be disseminated is of size 5 MB and is divided into 10 blocks, *i.e.*, $s = 500$ KB. We assume that a peer may leave the session once it completes its download, which is the usual behavior of peers in peer-to-peer networks. The source peer also leaves the session once its direct downstream peers complete the downloading process.

One of the advantages of using network coding is to use available bandwidth more efficiently. In Fig. 2, bandwidth efficiency is defined as the ratio between (1) the average number of bytes received by a peer before it can completely reconstruct the original file; and (2) the actual file size. The higher the ratio, the less bandwidth-efficient the algorithm is. We observe that network coding improves bandwidth efficiency by up to 20% (about 10% on average).

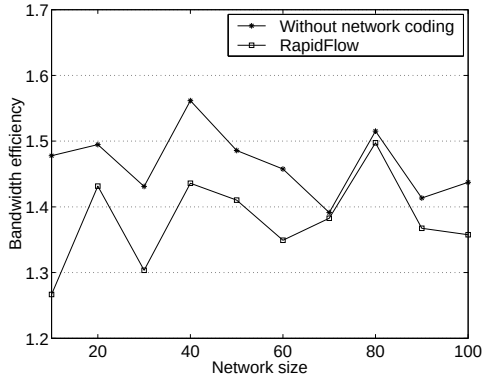

Fig. 2. Bandwidth efficiency of network coding.

However, we have observed that the encoding time introduced at each hop during transmission has aversely affected the total time to complete the download. Especially, peers farther away from the source experience longer downloading times than the ones close to the source. The downloading time records the duration from the first coded block was sent from the source, to the time the original file is completely reconstructed at a receiving peer, assuming that the coded blocks can be produced offline on the source. The average download time of all peers in the peer-to-peer session is presented in Fig. 3. It indicates that network coding leads to 100% longer download times in RapidFlow. We also observe that coding times increase as the number of blocks increases.

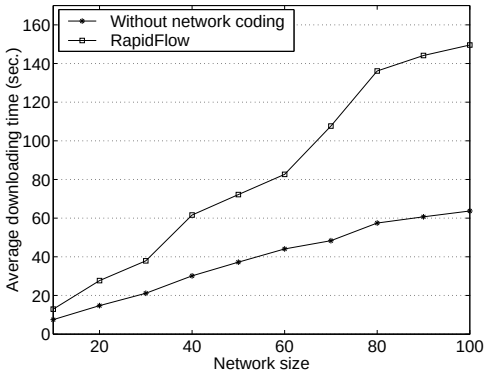

Fig. 3. Average times to complete the downloading process.

IV. Concluding Remarks

In this paper, we have presented *RapidFlow*, an implementation of network coding in peer-to-peer content dissemination sessions. RapidFlow represents our future research direction towards evaluating the advantages and drawbacks of network coding using a real-world implementation, rather than simulation-based studies. Our first batch of experiments have shown that, though network coding has led to more efficient use of bandwidth, it has significantly affected the time to complete the downloading process on each peer. We are still in the process of investigating the cause to this observation, and explore possibilities of improving our algorithms to mitigate such negative effects of using network coding.

References

[1] B. Cohen, "Incentives Build Robustness in BitTorrent," in *P2P Economics Workshop*, 2003.

[2] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proc. of the 24th Conference fo the IEEE Communications Society (INFOCOM'05)*, March 2005.

[3] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transaction on Information Theory*, vol. 46(4), pp. 1204–1216, July 2000.

[4] R. Koetter and M. Médard, "An Algebraic Approach to Network Coding," *IEEE/ACM Transaction on Networking*, vol. 11(5), pp. 782–795, October 2003.

[5] D. Lun, N. Ratnakar, R. Koetter, M. Médard, E. Ahmed, and H. Lee, "Achieving Minimum-Cost Multicast: A Decentralized Approach Based on Network Coding," in *Proc. of the 24th Conference fo the IEEE Communications Society (INFOCOM'05)*, March 2005.

[6] T. Ho, R. Koetter, M. Médard, D. R. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting," in *2003 IEEE International Symposium on Information Theory (ISIT)*, 2003.

[7] B. Li, J. Guo, and M. Wang, "iOverlay: A Lightweight Middleware Infrastructure for Overlay Application Implemenations," in *Proc. of the 5th ACM/IFIP/USENIX International Middleware Conference (Middleware'04)*, October 2004, pp. 135–154.