

Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 11-004

i-Code: A New Approach to Practical Network Coding for Content
Distribution

Hun Jeong Kang, Aaram Yun, Eugene Y. Vasserman, and Yongdae
Kim

February 18, 2011

i-Code: A New Approach to Practical Network Coding for Content Distribution

Hun Jeong Kang¹, Aaram Yun², Eugene Y. Vasserman³, Yongdae Kim¹

¹University of Minnesota, ² Ulsan National Institute of Science and Technology, ³ Kansas State University

Abstract—This paper studies the practicality of network coding to facilitate cooperative content distribution. Network coding is a new data transmission technique which allows any nodes in a network to encode and distribute data. It is a good solution offering reliability and efficiency in distributing content, but network coding has not been widely used because of its dubious performance gains and coding overhead in practice. With the implementation of network coding in a real-world application, this paper measures the performance and overhead of network coding for content distribution in practice. This study also provides a lightweight yet efficient encoding scheme which allows network coding to provide improved performance and robustness with negligible overhead.

I. INTRODUCTION

The continued rapid growth of computer networks, and the recent rise in the popularity of online entertainment products such as music rental and video-on-demand have strained existing content distribution infrastructure. Traditionally, content distribution systems are based on the client-server model, where clients download all requested content from dedicated servers. This model comes at a high cost in terms of server operation, but the advent of peer-to-peer (P2P) technologies has provided a new paradigm for content distribution. All content receivers (or nodes) also become content providers, cooperatively participating in the content distribution. Since their roles are equal in contrast to the asymmetric client-server relationship, participating nodes are called peers. P2P offers far better scalability properties for content distribution, at no cost to the distributor: instead of upgrading server and data center infrastructure as demand increases, peers participation naturally grows as demand for content increases, resulting in more peer-contributed resources, and providing a limited amount of “self-scaling.” Furthermore, P2P may be more robust to failure than client-server designs.

Despite their advantages and popularity, existing P2P content distribution systems suffer sub-optimal performance due to poor design, overly high peer turnover, and unforeseen emergent properties of large peer groups. In a typical cooperative content distribution system like BitTorrent [1], content is divided into blocks (or pieces). Peers exchange missing blocks with each other until they collect all component blocks and reconstruct the original content. As soon as a node acquires at least one block, it can offer it for download by others. This parallelizes downloads, such that peers can simultaneously download different blocks from different nodes, achieving higher throughput [1], [2]. However, this approach also poses significant challenges in the form of scheduling and

availability problems. Peers must make scheduling decisions about when to download particular blocks, and from whom. Peers also have a say in whether they will honor a given upload request by another peer. Since many systems reward uploading with tit-for-tat service (peers will preferentially upload to others from whom they have downloaded blocks in the past), upload scheduling adds another facet to the scheduling problem.

The two primary goals of a content distribution system are to minimize download time and bandwidth usage. Finding an optimal scheduling strategy that fulfils these requirements is difficult, especially when peers make local decisions without relying on central coordination. Consider the example in Figure 1(a), somewhat modified from its original formulation in [3]: peer B is about to complete a download of block X from peer A , and peer C needs to make a local decision regarding which blocks to download from A and B . If C decides to download X from A , then both B and C will have the same block X . This makes the link between B and C useless, since neither node has blocks that could help the other complete a download, and download time for both B and C will increase. To address the block scheduling problem, large-scale P2P systems use scheduling schemes such as random-first and local-rarest-first policies, but the resulting block scheduling is still often inefficient [1], [4].

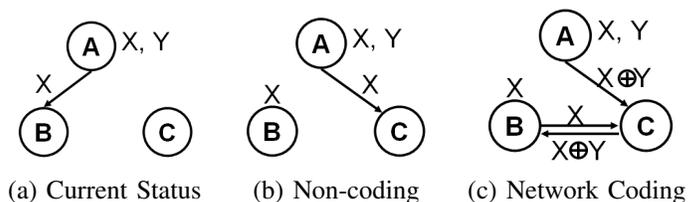
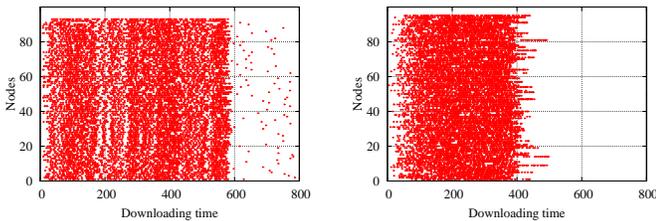


Fig. 1. A two-block example of the block scheduling problem

Furthermore, the differential availability of data blocks may affect the performance of content distribution systems. P2P networks are dynamic in nature — peers may arrive, depart, or fail frequently. This is referred to as peer dynamics, or churn. When some peers are not available, certain blocks may become rare. Peers missing rare blocks are forced to wait longer than others, since the queue to download rare blocks is longer than queues to download widely-available blocks. This issue makes efficient scheduling even harder. Worse yet, some blocks may be completely unavailable, as they are held by peers who happen to be offline. Since content cannot be

reconstructed when even one block is missing, some peers will fail to download the whole content due to a small portion of missing data.

Network coding has been considered as a potential solution to these problems [3], [5]–[12]. The key idea is to allow peers to “encode” their blocks when uploading them to other nodes. (We refer to the uploader as the “upstream” node, and the downloader as the “downstream” node.) In this paper, we focus on the popular random linear network coding [13] scheme, or more precisely some variants of it adapted for P2P systems. In this design, whenever a peer uploads a block to another node, it sends a linear combination of some or all of the blocks it has. This way peers no longer have to fetch a copy of each specific block, rather a peer simply asks another node to send a coded block, without specifying a block index. In Figure 1(b), peer C will download a single block from A , which is a linear combination of blocks X and Y . C no longer needs to concern himself with which blocks B will have. Then B and C can exchange blocks with each other, which efficiently uses the link between B and C and minimizes the downloading time. After a peer receives enough linearly independent blocks, he can reconstruct the original content, eliminating the requirement that each block be downloaded individually. Even when some peers with rare blocks leave a network, other nodes will not have difficulty in downloading coded blocks from remaining peers and recovering the original content, since rare blocks have been “mixed” into other blocks that remain in the network. Therefore, network coding can potentially provide better robustness and reliability for content distribution.



(a) Downloads in BitTorrent (b) Downloads in Network Coding

Fig. 2. Comparisons of downloads between BitTorrent and Network Coding

Consider the following experiment, in which we use BitTorrent¹ to distribute a 32MB file with 128 pieces to 95 nodes in PlanetLab [15]. All peers join the swarm at the same time and leave immediately after they finish their download. Their upload speed is limited to 100KB/s to better approximate consumer-grade network connections. In Figure 2, a point at (x,y) means that node y downloaded a piece x seconds after the start of the experiment. We observe many gaps in Figure 2(a), meaning there are many small time periods when nodes are forced to wait to download missing pieces from others, increasing their total download time. Some peers suffer especially long wait times toward the end of their downloads, when they have obtained most, but not all, of the pieces.

¹An unmodified CTorrent [14] client

Figure 2(b) demonstrates how network coding makes efficient content propagation easier. This experiment uses the same parameters as 2(a), except that modified BitTorrent clients use network coding.² The inherent benefit of network coding is that blocks do not have predetermined indexes, and so peers no longer have to fetch a copy of each specific block. Instead, they request the upstream peer to send a coded block that is a combination of blocks that peer already has. Adding network coding dramatically reduces periods of idle time.

Despite its benefits, network coding has not been widely used for content distribution, and there are doubts about the extend of performance gains achievable in practice [3], [11]. Network coding also introduces computational complexity and the excessive usage of resources such as memory and disk [6], [7], and different network coding schemes have different benefits and trade-offs in terms of computational overhead, wasted bandwidth, and download time. These factors together make real-world performance comparisons difficult.

A major issue is just how to perform data encoding to minimize encoding time, CPU overhead, and the number of independent blocks. In linear network coding, a node must decide how many blocks to combine every time it generates an outgoing block, since encoding time is not trivial. We measured an encoding time of 2 milliseconds to combine two 256KB blocks when using commodity hardware. When blocks are stored on disk, encoding time may increase significantly due to disk access delay, which varies depending on numerous factors such as disk speed, disk cache size, available system memory, and the number of page faults. We observed disk access times varying from 30 microseconds to 0.2 seconds to load 256KB of data. Random linear network coding was originally formulated such that *all* blocks available to a peer are to be combined to produce an encoded block [13]. In today’s content distribution, it is common that files are quite large and consist of hundreds or thousands of blocks (or smaller numbers of blocks but with larger block sizes). If we assume a block size of 256KB for those large files, it will take more than several seconds to encode a single outgoing block. Therefore, it is almost impossible to use this “full” encoding in practice. To reduce encoding overhead, a series of “sparse coding” schemes were introduced [6], [7], [10], [16]. These designs use fewer input blocks to generate a coded block, but result in more dependent blocks, especially when too few input blocks are combined. In linear network coding, the usefulness of data is determined by linear dependency. These dependent blocks do not contribute “useful” data to other nodes, since they carry duplicate information from other blocks, thus wasting bandwidth and time. High levels of block dependency delay content propagation, since peers have difficulty in locating independent blocks. There is a direct tradeoff between encoding overhead and block dependency, and until now there was no encoding scheme which achieved both low encoding overhead and low block dependency.

²We use i-Code (which is the primary contribution of this paper) for the network coding.

The primary contribution of our work is the design of *i-Code*, an encoding scheme satisfying both requirements of low encoding overhead and low levels of block dependency. *i-Code* combines only two blocks for every encoding operation, dramatically reducing encoding overhead. However, it does not have the dependent block penalty faced by other encoding schemes which combine few input blocks. The key idea is to emulate an encoding scheme which combines many input blocks. To that end, each peer using *i-Code* maintains a “well-mixed” block, which we call the *accumulation block* (\mathbf{a}). Whenever a peer receives an independent block (\mathbf{w}), it updates its accumulation block with the new data ($\mathbf{a} \leftarrow \alpha\mathbf{a} + \beta\mathbf{w}$, for randomly chosen coefficients α and β). When the peer encodes a new block, it selects a block from its local store and linearly combines it with the accumulation block. Therefore, all blocks the peer has are “accumulated” into \mathbf{a} . Because the accumulation block represents data from all current blocks of the peer, mixing any block with the accumulation block has a similar effect of combining many blocks. We compare the performance and overhead of each network coding scheme and show that *i-Code* exhibits the low level of block dependency comparable to the full coding with significantly less overhead and fewer dependent blocks than sparse coding schemes.

To attest to the practicality of *i-Code*, we also compare performance and overhead of BitTorrent, *i-Code*, and several other network coding schemes in real-world network environments. Prior studies on the benefits of network coding [3], [17] have been based on simulations or theoretical analysis, and may not reflect real network conditions. Although Gkantsidis et al. provide an implementation in [10], there is no real-world performance comparison between their network coding-enabled implementation and a non-coding system. There is more recent work showing potential practical benefits of using network coding for content distribution [6], [7], [11], but the experiments were performed with a small number of nodes or small files, and in network settings that were overly favorable to network coding, limiting the generalizability of the findings. With our *i-Code*-enabled BitTorrent client, we provide thorough empirical comparisons between BitTorrent and network-coded BitTorrent, using many nodes communicating over both a local and a wide-area network.³ Experimental results show that content distributing time of the coding-enabled “cP2P” system decreases by 5–21% compared to the P2P-only (BitTorrent alone) in real-world conditions.

This paper is organized as follows. Section II introduces the basic concepts and real-world performance issues and challenges in cooperative content distribution systems and linear network coding. Section III describes our practical network coding system with *i-Code* and Section IV provides the results of real-world tests of our implementation. Section V discusses related work and Section VI concludes the paper.

³We use PlanetLab [15] for wide-area network testing.

A. Cooperative Content Distribution

As a concrete real-world application for cooperative content distribution, we consider BitTorrent [1], the most popular P2P file-sharing protocol. It has been reported that BitTorrent traffic amounted to 30–80% of P2P traffic and 20–55% of all the Internet traffic as of 2008 and 2009 [18]. When a user wishes to share a file using BitTorrent, the client will break the file into smaller blocks, which can be downloaded independently in parallel. It then creates and publishes metadata (called a *torrent file*) which includes information such as the name of the content being shared, its total size, the hashes of the blocks, and the address of a *tracker*. The tracker is a central node that is responsible for helping peers find each other by keeping a list of peers participating in distribution of a given block of content (called a “swarm”), along with their IP addresses and downloading status. A user who wants to obtain the content first fetches the torrent file and contacts the tracker. During this bootstrap process, the tracker responds with a list containing a partial random subset of peers in the swarm associated with the content.

BitTorrent defines two types of peers: *seeders* and *leechers*. Seeders own a complete copy of the content and shares it, while leechers have either no blocks or an incomplete set, and are unable to reconstruct the content without additional blocks. Peers exchange `BitField` messages to determine which blocks the peers have. `BitField` is a series of bits mapped to blocks each peer has. Whenever a peer receives a new block it sends a `HAVE` message. Based on this information, a leecher knows who have its missing blocks, and ask them to upload. If none of the peers returned by the tracker have missing blocks, the peer can contact the tracker again, most likely getting a different list of peers in the swarm. For fairness, BitTorrent leechers prefer *tit-for-tat* exchange of content, typically choosing leechers to whom it will upload based on the amount of content downloaded from those peers in the past. This discourages “free-riding,” or when peers download content without uploading data to other nodes.

B. Random Linear Network Coding

Although network coding has been initially proposed to increase network throughput [19], [20], it can be used for improving the performance of content distribution system by simplifying the block scheduling problem. In this paper, we consider the popular linear network coding design [20], which is simple to implement and has been proven to achieve maximum throughput. More precisely, we consider variants of random linear network coding [13] adapted for P2P systems. In this scheme, a file is divided into m blocks, each represented as n elements in a finite field \mathbb{F}_p of size p , where p is prime. Then, i -th block can be considered as a vector $\tilde{\mathbf{u}}_i = (u_{i,1}, \dots, u_{i,n}) \in \mathbb{F}_p^n$ and the file becomes a sequence of vectors $(\tilde{\mathbf{u}}_1, \dots, \tilde{\mathbf{u}}_m)$.

When the original content source performs encoding with respect to a vector of coefficients $(\alpha_1, \dots, \alpha_m)$ (referred to as

the *encoding vector*), it computes an *information vector*, or a linear combination of $\tilde{\mathbf{u}}_1, \dots, \tilde{\mathbf{u}}_m$,

$$\tilde{\mathbf{w}} = \sum_{i=1}^m \alpha_i \tilde{\mathbf{u}}_i = (w_1, \dots, w_n). \quad (1)$$

(The choice of encoding vectors depends on the type of coding, and can be a global parameter. In random linear network coding [13], each node independently chooses encoding vectors randomly.) The source then sends the encoding and information vectors together in the *augmented block* (or simply *block*) with the augmented form of $\mathbf{w} = (\alpha_1, \dots, \alpha_m, w_1, \dots, w_n)$. When uploading an augmented block, a peer sends a linear combination $\sum_j \beta_j \mathbf{w}_j$ of its received blocks $\mathbf{w}_1, \dots, \mathbf{w}_l$. A node can decode the original file after receiving at least m linearly independent blocks. If W is the information vectors of received blocks and A is the matrix whose rows are the encoding vectors of received blocks, the receiver can recover all original blocks of file U by solving the linear equation $W = AU$.

C. Tradeoff in Network Coding

With the use of network coding, we must decide how to encode data at each node in a network. There are many ways in which nodes can select and combine received blocks $\mathbf{w}_1, \dots, \mathbf{w}_l$ to produce an outgoing block $\sum_j \beta_j \mathbf{w}_j$. In addressing this issue, we mainly consider two criteria: encoding overhead and block dependency.

One obvious method would be to use all blocks in a node's local store and combine them using coefficients β_j chosen uniformly at random from the field \mathbb{F}_p . This is the approach taken in traditional random linear network coding [13], which we will call *full coding* (since all blocks are used for each encode operation) in order to distinguish it from other schemes. In [21], Yeung showed that full random linear network coding used for P2P systems achieves maximum possible throughput. As previously mentioned, however, it is impractical to use full coding at "line speed," where a node would produce an encoded block in real time, when requested by another node, due to CPU and disk overhead.

To reduce the encoding time, peers may generate a coded block using fewer input blocks. The encoding overhead can also be ameliorated by splitting the file into a small number of *generations*, and using full coding on each generation as opposed to the entire file, reducing the number of blocks that are encoded for each request. We call this scheme *gen coding*. Another way to use fewer blocks is the *sparse random linear network coding* (or *k-coding*), where a node encodes up to k randomly-selected local blocks $\mathbf{w}_{i_1}, \dots, \mathbf{w}_{i_k}$, forming a random linear combination $\sum_{j=1}^k \beta_j \mathbf{w}_{i_j}$. Unfortunately, sparse coding is significantly less efficient than full coding. Consider the scenario in which a node, having already received m' independent blocks, has just received a new independent block \mathbf{w} . This node has new information about the file (namely \mathbf{w}), but an outgoing block from the node would contain \mathbf{w} in its linear combination with probability $k/(m' + 1)$. Initially, when m' is small, the new information \mathbf{w} can be propagated

through outgoing blocks with high probability, but as m' grows, especially when $m' \approx m$, this probability is only $\approx k/m$. There is a clear trade-off between the value of k (and thus CPU overhead) and the bandwidth utilization (goodput) in the network. Setting k to be too small generates unnecessary dependent blocks, and setting a large k reduces the benefits of *k-coding*.

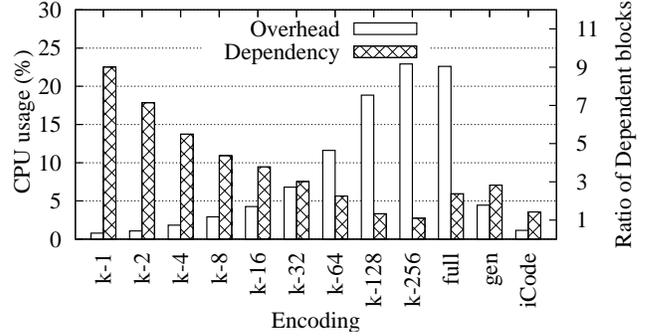


Fig. 3. Tradeoff between CPU overhead and block dependency

Is there an encoding scheme that offers both low encoding overhead and provides has few dependent blocks? Figure 3 demonstrates the trade-offs between these requirements by comparing full coding, gen coding, and sparse coding with various values of k . White bars represent CPU usage and crosshatched bars represent relative amounts of dependent blocks when compared with optimal coding. Note that coding with small k generates many dependent blocks, and large k as well as full coding use more CPU. (This graph is explained in more detail in Section IV.) Our novel network coding scheme design, *i-Code*, which combines the benefits of sparse coding (low overhead) and full coding (few dependent blocks), is also included in this graph. We describe this lightweight and efficient encoding scheme in the next section.

III. I-CODE

We aim to provide a practical network coding scheme which can be used in real-world content distribution systems, but there is a trade-off between block dependency and overhead for encoding. No existing scheme fulfills both requirements of low block dependency and low encoding overhead. Therefore, we propose a new encoding scheme which we call *i-Code*, the contraction of "incremental encoding." The key insight is to "emulate" the full coding with a small number of linear block combinations. For this, a peer maintains a pre-encoded "well-mixed" *accumulation block* which contains a linear combination of blocks which a peer already has. When encoding a new output block, the peer selects a single locally-stored block and combines it with the accumulation block. *i-Code* is light-weight and performance-efficient, and produces few dependent blocks while maintaining low encoding overhead.

The pseudo-code for accumulation block maintenance is in Algorithm 1. Consider peer A which is receiving blocks from peer C and is sending encoded data to peer B . (Nodes

Algorithm 1 i-Code operations

```
1: Parameters: Accumulation block  $\mathbf{z}$ , bit-field  $bf_{accumulated}$ 
2: procedure IN( $\mathbf{x}$ ,  $bf_{sender}$ )
    $\triangleright$  Incoming block  $\mathbf{x}$ , bit-field for sender  $bf_{sender}$ 
3:   if  $\mathbf{x}$  is independent then
4:     randomly select coefficients  $\alpha, \beta$ 
5:      $\mathbf{z} \leftarrow \alpha\mathbf{z} + \beta\mathbf{x}$ 
6:     store  $\mathbf{z}$  into local store
7:     set  $bf_{accumulated}(x)$   $\triangleright$  record  $\mathbf{x}$  is mixed with  $\mathbf{z}$ 
8:     set  $bf_{sender}(x)$   $\triangleright$  record  $\mathbf{x}$  is shared with sender
9:   else
10:    discard  $\mathbf{x}$   $\triangleright$  a useless block
11:   end if
12: end procedure
13: procedure OUT( $bf_{receiver}$ )  $\triangleright$  bit-field for receiver
14:   Outgoing block  $\mathbf{w}_o$ 
15:   select a block  $\mathbf{y}$  from local store
16:   randomly select coefficients  $\gamma, \delta$ 
17:    $\mathbf{w}_o \leftarrow \gamma\mathbf{z} + \delta\mathbf{x}$   $\triangleright$  encode the outgoing block
18:   randomly select coefficients  $\gamma', \delta'$ 
19:    $\mathbf{z} \leftarrow \gamma'\mathbf{z} + \delta'\mathbf{x}$ 
20:   set  $bf_{accumulated}(x)$   $\triangleright$  record  $\mathbf{y}$  is mixed with  $\mathbf{z}$ 
21:   set  $bf_{receiver}(y)$   $\triangleright$  record  $\mathbf{y}$  is shared with receiver
22: end procedure
```

with whom A communicates are A 's "neighbors.") When A receives a block (\mathbf{x}) (lines 2–12), it first verifies if the incoming block is not linearly dependent on blocks that A already has (line 3); linearly dependent blocks are dropped (line 10). To reduce bandwidth consumption wasted by dependent blocks, A receives encoding vectors first. If the block is independent, the accumulation block \mathbf{z} is updated by $\mathbf{z} \leftarrow \alpha\mathbf{z} + \beta\mathbf{x}$, with randomly chosen coefficients $\alpha, \beta \in \mathbb{F}_p$ (lines 4 and 5). Then \mathbf{x} is stored on disk. We assume that the majority of blocks a peer has will reside on disk, and must be read into memory as needed, since in modern P2P networks, multi-gigabyte files are common and peers may not have enough memory to cache its all blocks in RAM. Even though some nodes may have enough memory, small memory usage is a desirable feature of any software system. When sending an encoded block \mathbf{w}_o (lines 13–22), A selects a single block \mathbf{y} from disk (line 15) and computes $\gamma\mathbf{z} + \delta\mathbf{y}$, a random linear combination of the accumulation block and \mathbf{y} (lines 17 and 18). It also updates \mathbf{z} similarly as a random linear combination with \mathbf{y} (i.e., $\mathbf{z} \leftarrow \gamma'\mathbf{z} + \delta'\mathbf{y}$) (lines 18 and 19). In this way, all incoming blocks and stored blocks can be accumulated to \mathbf{z} . Mathematically, the accumulation block is a "generic" random block of the subspace spanned by the peer's blocks.

To choose blocks from the disk (line 15), i-Code uses heuristics to reduce the probability of generating dependent blocks. Suppose that peer A is sending blocks to other nodes, including peer B . First, A chooses a block which is *not*

accumulated to \mathbf{z} . When such a block is linearly combined with the accumulation block for encoding, the outgoing block is likely to be independent to blocks A previously generated, or blocks B already has. To record whether a block is stored in the disk and is accumulated to \mathbf{z} , i-Code uses a small data structure, similar to the bit-field in BitTorrent, imposing almost negligible storage overhead. Second, the choice of \mathbf{y} depends on the recipient B and the transfer history: A chooses a block (\mathbf{y}) which has not been received from B , or has not been already used to produce an output block which has been sent to B . This information is also stored in a per-neighbor bit-field. BitTorrent clients (which do not use network coding) also maintain bit-fields to record which blocks neighbors have, so i-Code keeps roughly the same amount of in-memory state as BitTorrent clients.

i-Code has several advantages. First, it has very low computational overhead: for each encoding operation, only two blocks need to be mixed, and only one of those is read from disk. i-Code is also efficient: although this scheme combines a small number of blocks, it does not impose the dependent block penalty faced by the sparse coding. Because many blocks are already mixed into the accumulation block, mixing these two blocks has similar effects to full coding, or combining the most blocks in the peer's disk. Compared to sparse coding, i-Code has even further benefits. Suppose again that peer A receives a new independent block \mathbf{x} . In sparse k -coding, A randomly selects k blocks, and \mathbf{x} may not be included in new outgoing blocks. This means that A might encode the outgoing block from its blocks which were already mixed into previous outgoing blocks. This produces dependent blocks with a higher probability than the case in which the new independent block is mixed for the outgoing block. However, i-Code allows a newly-received block to be immediately combined with any outgoing blocks, and reduces the probability of encoding dependent blocks.

Despite these benefits, i-Code may still generate more dependent blocks than the full coding. Let V_A and V_B denote subspaces spanned by blocks A and B have, respectively. With i-Code, A sends a dependent block to B only when two unlikely conditions are met concurrently: i) after A sends a block to B , no new independent block is combined with the accumulation block and ii) A picks a block to upload which is in V_B . This is very rare in real-world applications, where each peer is continuously downloading blocks from many neighbors at the same time. Furthermore, A reduces the probability of selecting a block which belongs to V_B by referencing its block transmission history (i.e. bit-field information). In practice, i-Code produces very low levels of dependent blocks. Real-world measurements (shown in the next section) demonstrate dependent block levels comparable to full coding.

IV. EVALUATION

To evaluate performance of various content distribution systems in a real-world application, we integrated various

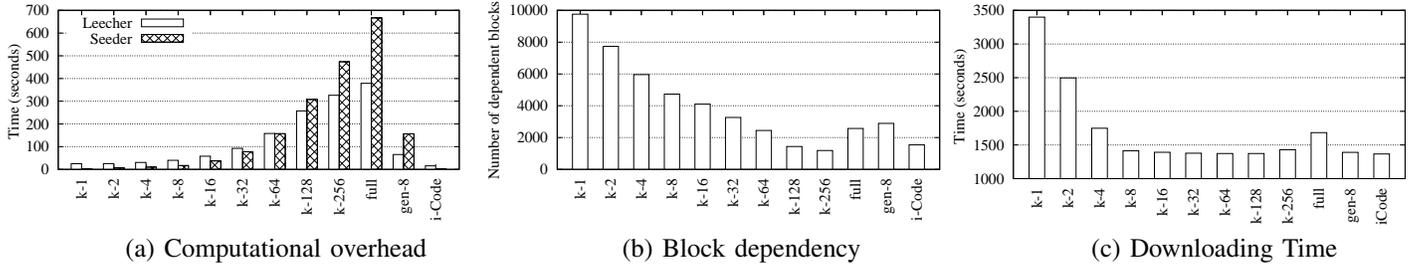


Fig. 4. Comparison of coding schemes with simultaneous joining

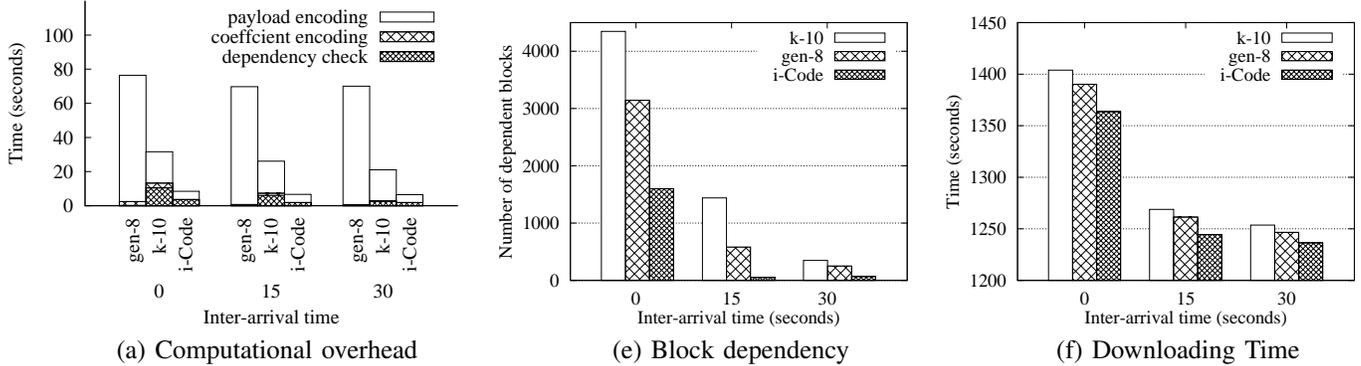


Fig. 5. Comparison of coding schemes with inter-arrival times (local testbed)

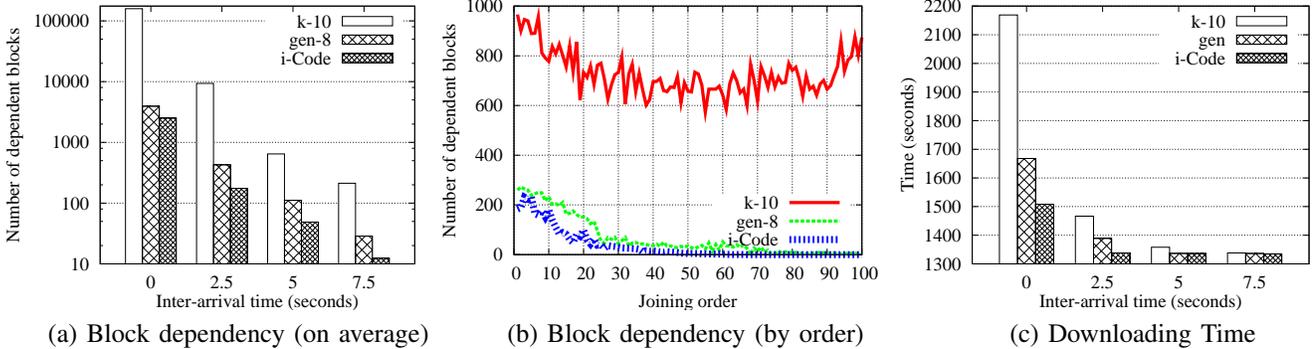


Fig. 6. Comparison of coding schemes with inter-arrival times (PlanetLab)

network coding designs into Enhanced CTorrent⁴. This system allows us to measure the performance and overhead of BitTorrent as well as multiple network coding schemes in the same environment, ensuring a fair comparison. First, we demonstrate the effectiveness of i-Code in achieving low encoding overhead and the low levels of block dependency, outperforming other encoding schemes. We then compare i-Code-enabled BitTorrent with vanilla BitTorrent using multiple variables, many that are often overlooked in other comparative studies.

We performed our experiments in two different networks: our local campus network and the PlanetLab [15] wide-area network testbed. The local network is composed of 25 nodes, each equipped with a 1 GHz dual-core AMD Opterontm processor and 2GB DDR2 RAM. The network is fast switched 100TX full-duplex Ethernet with low latency, high throughput, and negligible packet loss. The experiments conducted in this

testbed demonstrate performance in idealized network conditions, without interferences from network dynamics such as highly-variable latency, asymmetric and inconsistent connectivity, and bursts of packet loss that are often seen in Internet-connected wide-area systems such as PlanetLab. Furthermore, PlanetLab nodes are more resource constrained and often unreliable. These settings better reflect conditions encountered by real-world content distribution systems.

A. Comparison with other encoding schemes

We compare i-Code with full coding, sparse coding, and *gen* coding introduced in Section II. We evaluate the performance of each encoding scheme by measuring computational overhead (time spent on encoding and checking dependency), level of linear dependency (the number of linearly dependent

⁴A simple BitTorrent client written in C++ [14]

blocks⁵), and total time required to complete each download.

To make our results comparable with previous studies, we use a similar experimental setup to [6]: nodes are constrained to 100KB/s upload bandwidth, using 256KB piece size. Each node is connected to at most 50 neighbors. In each experiment, a seeder distributes a 128MB file consisting of 512 blocks to other peers. For more fine-grained analysis, we chose schemes shown to perform well prior studies: *gen-8* [7] and *k-10* [6].

1) *Simultaneous join*: We first conducted experiments with a simple scenario: all nodes join a content distribution session simultaneously, and leave immediately after completing their download. This is classic “flash crowd” behavior — many peers join simultaneously, putting great strain on the network to accommodate many new peers with no blocks to offer. Furthermore, all nodes are selfish, and do not contribute bandwidth to the network after completing their own download — they only share when forced to share, that is, in order to obtain the content.

Figure 4(a) shows computational overhead in this scenario, measured in our local low-latency testbed. Y-axis represents the average of total computation time at each node during the content distribution session. In sparse coding schemes, which are denoted as *k-x*, computational overhead increases as *x* increases. Overhead introduced by full coding is especially heavy: whenever a node sends a block, it must combine 512 blocks, spending a total of 665 seconds of wall-clock time on encoding and thus consuming 32% of CPU cycles on average.

We also measured the block dependency by counting the number of dependent blocks each peer received. In Figure 4(b), as the number of combined blocks decreases, the number of received dependent blocks increases. However, full coding is a notable exception to this pattern: the number of dependent blocks is higher than some sparse codings, such as only mixing 64 blocks. This is because a seeder cannot provide fresh blocks in real time, and, therefore, the number of independent blocks that leechers have increases slowly. Thus, leechers try to download “stale” blocks, producing more dependent blocks.

Figure 4(c) shows the time to download 512 independent blocks in each encoding scheme. Both computational overhead and linear dependency affect the downloading time. Generally, as a peer combines more blocks (i.e., *x* in *k-x* increases), the downloading time decreases because a peer is more likely to receive independent blocks. Full coding is an exception, as well as sparse codings with large numbers of combined blocks (i.e., large *x* in *k-x*), which result in longer download times due to excessive computational overhead.

From the results in Figure 4(a) and (b), we see that there is a direct trade-off between the number of dependent blocks and computational overhead: sparse and full coding either produce many dependent blocks or require heavy computation. i-Code, on the other hand, offers both small numbers of

dependent blocks and small encoding overhead. Moreover, i-Code provides faster downloading compared to other schemes as shown in Figure 4(c).

2) *Joining with intervals*: We also conducted experiments in which the times when nodes join a content distribution session more slowly, instead of all joining at once. Like in previous experiments, nodes leave the network as soon as they finish downloading. Since i-Code has encoding time comparable to combining approximately two blocks (the accumulation block and another block from disk), encoding overhead of an i-Code seeder is $\frac{1}{64}$ of *gen-8* coding and $\frac{1}{10}$ of *k-10* coding. In practice, however, leechers do not have all blocks, so the number of blocks to be mixed for an encoding block changes depending on the number of blocks at each leecher, so we compare the computational overhead of each coding scheme based on actual measured results. Figure 5(a) presents average computational overhead at each leecher, and how it is affected by the join interval. This overhead includes i) time to generate encoding vectors (coefficient encoding in the figure), ii) time to encode information vectors (payload encoding), and iii) time to check the linear dependency of received blocks (dependency check). A leecher using i-Code suffers the least overall computational overhead — 10% of *gen-8* coding and 26% of *k-10* coding, on average, with any inter-arrival times. *gen-8* coding has lower overhead in checking block linearity. Block dependency is determined by performing Gaussian elimination, with overhead of $O(m^2 \times n)$, where *m* is the number of blocks and *n* is the dimension of a block. By using multiple generations, *gen* coding can reduce *m*, resulting in reduced overhead for dependency checking. However, the time to combine information vectors dominates other overhead, and i-Code performs this operation much faster. Figure 5(b) shows the number of received dependent blocks according to inter-arrival times. i-Code produces fewer dependent blocks than *k-10* and *gen-8* encoding schemes regardless of joining intervals. From Figure 5(c), we see that clients using i-Code complete downloads faster than other schemes with any inter-arrival time. These figures clearly show i-Code has better performance and smaller overhead.

In Figure 5, we observe that peers has less dependent blocks, smaller overhead, and shorter downloading time as the inter-arrival time becomes longer. We can explain this observation using the concept of vector spaces. Let subspace spanned by blocks which a peer has be simply the peer’s subspace. In flash-crowd (i.e., the simultaneous joining), most nodes in the network are likely to have similar subspace with same (or similar) dimensions. Only a content source has seriously different subspace and sends out new linearly independent blocks to the network and these blocks are propagated into the nodes in the network while being mixed with other blocks. Except these blocks, most of blocks transmitted between peers are dependent with high probability. This is why the flash-crowd produces many dependent blocks. However, when peers join the network with intervals, peers have diverse subspaces depending on their joining time neighbors. Therefore, peers have more chances to receive independent blocks from several

⁵When a i-Code-enabled BitTorrent client receives a block, it first accepts only an encoding vector of the block and checks if it is linearly independent. If it is independent, the client downloads the remaining block. If not, the client discards the encoding vector. For simplicity, we use dependent encoding vectors and dependent blocks interchangeably in this section.

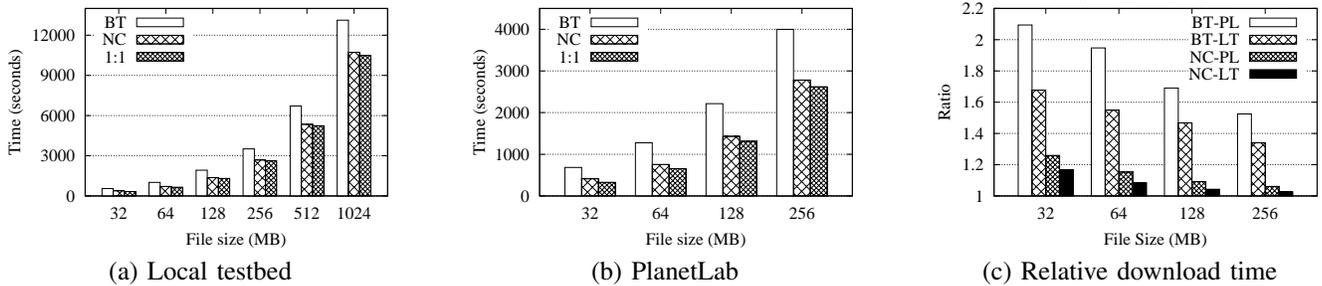


Fig. 7. Comparisons of downloading with simultaneous node joining (*setup1*)

uploaders. When the joining interval is larger, peers have more diverse subspace and receive less dependent blocks. With less dependent blocks, peers have smaller overhead for linear dependence check and coefficient encoding. The low level of block dependency also leads to shorter downloading time.

3) *Swarm size*: We have shown the experimental results with a small number of nodes. We now present experimental measured in the PlanetLab environment with 100 nodes under the same scenarios used for experiments in the local testbed. We do not provide the computation overhead because PlanetLab nodes have different system configurations and measurement results may include noises such as CPU consumption by other processes.

Like the results of experiments conducted in the local testbed, i-Code produces less dependent blocks than other schemes in Figure 6(a). Compared to Figure 5(a), the absolute number of dependent blocks of each schemes becomes larger because each peer is connected to more neighbors and exchange more blocks with other nodes. However, notice that i-Code produce relatively much fewer dependent blocks than other schemes in Figure 6(a). k -10 codes produces 12.5 times more dependent blocks than those of i-Code in the local testbed experiments but k -10 codes produces about 40.3 times more dependent blocks than i-Code in the PlanetLab tests.

i-Code generates almost no dependent blocks when nodes join the network incrementally, without forming a flash crowd. Figure 6(b) compares the numbers of dependent encoding vectors when peers joined every 5 seconds. Peers using i-Code do not generate dependent blocks except when they join the network early in a content distribution session, when a seeder has not yet had a chance to transmit many blocks into the network (i.e. there are not many independent blocks in the network). Accordingly, peers are likely to download dependent blocks at a rate similar to simultaneous joining. However, peers using k -10 produces 600 – 1000 dependent encoding vectors regardless of time when they join the network in the figure. From Figure 6(c), we again see that i-Code provides shorter downloading time than other encoding schemes due to low block dependency.

In summary, i-Code incurs encoding overhead equivalent to k -2 coding, but with block dependency levels close to k -128 encoding. With this smaller overhead, i-Code enables

faster download than all other encoding schemes, as shown in Figure 5(c).

B. Network coding in real networks

To evaluate network coding for P2P content distribution in a real-world application, we compare a non-coding system and a network coding-enabled system. We again use the Enhanced CTorrent BitTorrent client as our software framework; we call our i-Code-enabled version *iTorrent*. For simplicity, we refer to the former as *BT* and the latter as *NC*.

1) *Performance comparison*: The most natural metric of content distribution performance is required time to download a single complete file. We measured performance of NC and BT using two sets of experimental variables: one for comparing our results with previous studies [6], [7] (*setup1*), and the other for evaluating content distribution methods for practicality and performance in real-world environments (*setup2*). We note that the two setups differ from each other significantly — one of our contributions is evaluation of i-Code as well as other coding methods using the realistic *setup2*.

Simultaneous joining with *setup1*: In this experiment, a single source distributed a file with block size was 256KB. Each node was connected to at most 50 neighbors and their upload bandwidth was limited to 100KB/s. Peers joined a distribution session at the same time and left the session immediately after completing the download. Figures 7(a)-(b) show the times required to complete a download. In addition to BT and NC, we also show “1:1” downloading as an ideal case, in which a client directly downloads content from a dedicated server (without using P2P). Overall, NC reduces the download time by about 18 – 40% compared to BT. As described in Section I, finding the optimal data propagation scheduling (that minimizes downloading time) in a distributed way is very difficult. However, the result shows that NC provides near-optimal downloading time, especially for larger files. This occurs because in the case of “flash crowds,” when many nodes join the session at the same time, most nodes will end up waiting to download their first few blocks, but the proportion of this initial waiting time to the downloading time becomes smaller as the content size grows. Figure 7(c) compares downloading times measured in both experimental networks. “LT” represents experiments on our local-area network, and

“PL” represents the PlanetLab wide-area testbed [15]. The Y-axis is the ratio of measured download times to 1:1 time. Notice that the difference between downloading times of BT and NC is larger in PlanetLab, where nodes exhibit more dynamic and unexpected behavior.

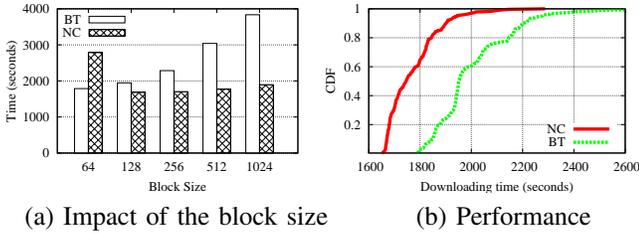


Fig. 8. Performance comparison of simultaneous joining (*setup2*)

Simultaneous joining with *setup2*: To measure our system under more realistic settings, we used 100 PlanetLab nodes, with upload bandwidth capped at 80KB/s (the median upload capacity measured in [22]). Each peer was connected to at most 50 neighbor nodes, the default value in CTorrent [14]. The file size is 128MB. To determine the appropriate block size, we measured how block size affects performance of BT and NC. In our experiments, peers joined a content distribution session at the same time and left immediately after completing the download. We varied the block size from 32KB to 1MB at each experiment, although block sizes less than 64KB are not used due to excessive message overhead [1], [23]. Figure 8(a) shows the downloading times as a function of block size. As the block size increases, the downloading time increases in both BT and NC because large blocks are propagated slowly and each peer has difficulty in getting the first several blocks at the beginning of a content distribution session. One exception is NC with block size of 64KB, where we see longer downloading time because of excessive computational overhead: if the block size is small, the sizes of encoding vectors will be comparatively large, leading to more message overhead. In addition, small block size also introduces a large coefficient matrix and thus increased overhead during the dependency check. Overall, NC has the shortest downloading time when using 128KB blocks, while BT’s shortest download time is at 64KB. From these observations, 64KB and 128KB are used for the block sizes of BT and NC, respectively. In this new setup, NC reduces the downloading time about 10% compared to BT (Figure 8(b)). NC does not provide as big a performance improvement in *setup2* as in *setup1*. However, we believe this setup is fair to both systems, and more representative of real-world applications. Thus we will use *setup2* for the following experiments.

Dynamic joining and seeding: We also investigated the performance impact of peer arrival and departure times. In Figure 9(a), peers arrived at different times and left the session as soon as they finished downloading, with the X-axis showing the time between peer arrivals. Network coding provides shorter downloading time, but the improvement decreases as the inter-arrival time increases: peers could get data relatively

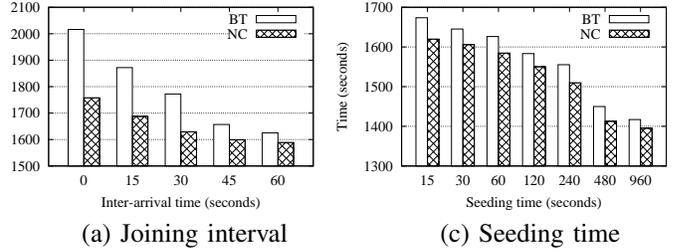


Fig. 9. Downloading times according to peer arrivals and departures

easily because there were already enough blocks in the network when peers joined at different times. In Figure 9(b), peers arrived every 15 seconds and stayed for a given seeding time after completing their download. When there are many seeders, peers can easily download from those seeders and were less likely to suffer from the scheduling problem. NC provided an improvement of 1.5 – 3.2%, and the performance difference between two systems decreased as the seeding time increased.

2) Robustness comparison: In some situations, BT may suffer robustness problems which increase download time: during a distribution session, some content blocks may become rare, and peers which are missing the blocks need to wait for a disproportionately long time to receive those blocks. Even worse, the rare blocks may disappear altogether as the few peers who have them leave the network, stopping others in the content distribution session from finishing their download. This problem may occur because of uneven data propagation and peer dynamics, including the actions of the original source. We compare robustness of BT and NC by examining the download time and percent of peers who completed their download. We varied the times nodes stayed in the network, the number of neighbors from which a peer can download blocks, and nodes with differing available resources.

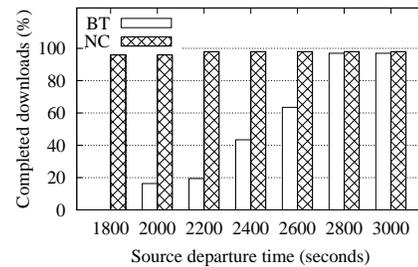


Fig. 10. Completeness according to the time of source departure times

Node departure: To introduce the rare block problem into the content distribution session, we conducted experiments similar to those in [3], [6]. The source sent out data for a given amount of time and then left the session. Peers arrived every 30 seconds and left the immediately after finishing their download. Figure 10 shows the fraction of nodes which completed their download, as a function of the time the original file source stays in the network. Recall that it took 1638 seconds, on average, to send the entire content at 80KB/s.

When the source leaves the session at 1800 seconds, it is enough time to send the entire content to nodes in the network; remaining nodes should be able to exchange missing blocks to complete their downloads. However, experiments show that no nodes managed to finish downloading the entire content. When the source transmits data for a longer time, this problem gradually fades. On the other hand, network coding systems do not rely on specific blocks of data, and therefore do not exhibit this problem — completion rates remain at a constant 95%, independent of the actions of the source.

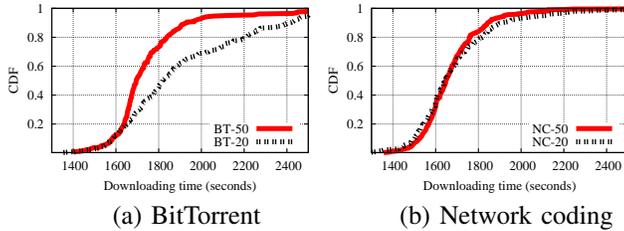


Fig. 11. Impact of number of neighbors

Limited number of neighbors: We also examined the impact the number of neighbors connected to each peer. In a distribution session, peers form an overlay network and exchange information on peers or data pieces which they have. If a peer is connected to too many neighbors, it becomes too resource-consuming to maintain connections and to exchange messages such as “keep-alive” and bit-field-like information. Thus, peers commonly limit their number of connected neighbors. In [3], [6], [7] each peer is connected to at most 10 – 20 neighbors while the default numbers of maximum neighbors in CTorrent and Mainline5.3 [14], [24] are 50 and 80, respectively. Figure 11 shows the download completion times with different maximum numbers of neighbors. Each peer in BT was connected to at most 50 neighbors in the “BT-50” experiments, and 20 neighbors in the “BT-20” experiments. Since nodes with only 20 neighbors have less information about blocks other peers have in the network, they face difficulties in performing block scheduling, and consequently exhibit greater download times than BT-50. However, network coding does not suffer from the block scheduling problem to the same extent, since it does not have to schedule which specific blocks to download. NC showed shorter downloading times compared to BT regardless of the number of neighbors.

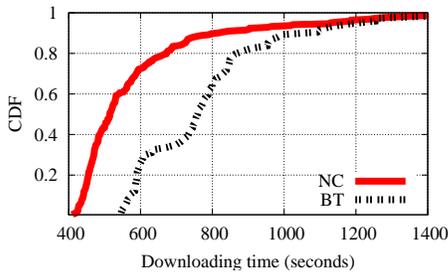


Fig. 12. Downloading times of fast nodes in heterogeneous environments

Heterogeneous capacities: Nodes generally have non-homogeneous capabilities. We studied the performance of content distribution systems in heterogeneous environments. We separated 200 PlanetLab nodes into two groups: 100 fast nodes with a 420KB/s maximum upload rate and a 4MB/s maximum download rate, and 100 slow nodes with an 80KB/s maximum upload rate and a 1MB/s maximum download rate. A source belonging to the fast group distributed a 128MB file to 200 nodes. Figure 12 shows the downloading times experienced by fast peers when they interact with other slow nodes. With NC, the download times of the fast nodes were 37% better than BT on average. Without network coding, it takes a long time for fast nodes to download their missing blocks, especially when slow nodes have those blocks. With network coding, on the other hand, blocks that propagate through the network are linear combinations of many other blocks, so fast nodes have more chances to find useful blocks from other fast neighbors, and can make quick download progress.

3) *Overhead Comparison:* We have seen that network coding provided better performance and robustness. We examined how much network coding introduces overhead in order to provide for the improvement.

Encoding overhead: Recall again that it takes 1638 seconds to download 128MB content from other nodes with an 80KB/s upload rate using network coding in ideal 1:1 coding. The total average time to encode outgoing blocks in each peer was less than 10 seconds (less than 1% of the downloading time) including the disk access time. Message overhead for delivering coefficient vectors was about 1MB, 0.78% of the content. When peers joined the session every 15 seconds, the additional average message overhead for delivering dependent encoding vectors was only about 6510 bytes in each node. However, NC saved far more bandwidth than BT. Recall the 128MB file was divided into 2048 and 1024 blocks in BT and NC, respectively. More blocks, more message overhead because each client sent its neighbors HAVE messages notifying the client received a new block. A BT client consumed 8.67MB for exchanging bitfields while a NC client spent 4.54MB, on average. As a result, NC introduced smaller message overhead than BT with affordable encoding overhead.

Decoding overhead: We have not counted decoding time for the decoding time in this paper. We now briefly discuss decoding process and its overhead. The complexity of the decoding process is $O(m^2n)$ where m is the number of blocks and n is the number of elements in an information vector. Here we use $O()$ notation to hide constants. For a 128MB file with 512 blocks (i.e., 128KB block size), it took 280 seconds to decode the file. To reduce the decoding time, we used multiple generations because the use of g generation reduces the decoding complexity to $O(m^2n/g)$. When we used 4 generations, it took 35 seconds to decode a 32MB generation and it took 140 seconds to decode the entire file. Furthermore, once a node downloaded one generation, it could start decoding the generation while receiving blocks in other generations. Thus, the node could finish decoding 35 seconds after downloading

the last generation. Using multiple generations might introduce the *generation scheduling problem*. However, the use of a small number of generations did not significantly increase the downloading time. For a 128MB file distribution, the delay due to multiple generations was less than 10 seconds when we used i-Code with 4 generations. Because of space limitation, we did not include the results in this paper.

4) *Summary of comparisons*: Network coding could provide near-optimal block scheduling. Even in a flash-crowd without seeding, the near-optimal scheduling enabled peers to download the content almost at the ideal finish time. In addition, network coding was robust to environments where BT suffered from the performance degradation. In PlanetLab where nodes showed more dynamic and unexpected behavior, NC provided more reliable performance in the sense that the difference between downloading times of BT and NC was logner in such hostile environments. When BT peers had a small number of neighbors, they had limited information on which blocks other nodes had. This led to inefficient block scheduling and thus slowed downloading time. However, NC still showed better performance even though peers had a small number of neighbors by virtue of near-optimal scheduling. Similarly, NC provided a stable and shorter downloading time with varying block sizes while BT's downloading time was relatively longer and largely affected by the block size. In heterogeneous environments, NC nodes with a fast link speed do not experience delayed downloads compared to BT.

Due to the use of i-Code, encoding time could be reduced to less than 1% of the downloading time without sacrificing performance. Message overhead for delivering coefficient vectors was also less than 1% of the content size. The overhead for exchanging bitfield-like information was 3.5% of the content size which was half of the overhead of BT. Therefore, NC had less message overhead than BT in total. Furthermore, a node could largely hide the time delay for decoding by progressively decoding received generations. In sum, network coding could provide improved performance and robustness with affordable overhead.

V. RELATED WORK

To improve distribution speed and resolve data availability problem, many state-of-art P2P content distribution systems such as BitTorrent [1] use local-rarest-first or random block scheduling policies. Additionally, a number of P2P systems have proposed the use of source coding with erasure codes to efficiently transfer bulk data [25], [26]. However, these two classes of approaches still suffer inefficiency and reliability problems [3]. Network coding was originally proposed to improve network throughput in a given topology [19], but there is now a large body of work on applying network coding, especially random linear network coding [13], to cooperative content distribution.

Many studies have explored the usefulness of network coding for P2P content distribution. Most, such as [3], were largely based on simulation or theoretical analysis, and may not generalize to real networks. Based on [3], Gkantsidis et al.

implemented a prototype network coding system for content distribution system and tested it for distributing large files [10]. While this demonstrated feasibility of using network coding in P2P systems, the authors did not compare the performance of their system with previous P2P content distribution schemes such as BitTorrent. Wang and Li [11] performed just such a comparison by modifying an existing P2P design to use sparse random linear network coding, using a cluster of high-performance servers with emulated bandwidth restriction for the experimental testbed. However, a small number of nodes and small file sizes may mean that their evaluation still does not generalize to real-world networks with complex peer dynamics, strange connectivity issues, and large file sizes.

Motivated by [3] and [10], Yeung [21] studied the use of full random linear network coding in P2P networks, which he modeled as time-expanded trellis networks, where a physical node x at time t is represented as a node (x, t) . Under this representation, a network-coded P2P system becomes an acyclic directed graph, and full random linear network coding achieves the maximum possible throughput. In [8], Yang et al. studied P2P file sharing using deterministic network coding with highly structured topologies, in contrast to the usual use of random linear coding and unstructured topologies. Results showed better performance, but managing the structured topology still makes this design more difficult in practice than using random coding.

Random linear network coding was originally formulated such that *all* pieces in a peer's local store were combined to produce an encoded block [13]. However, this approach is not practical due to its encoding overhead. To reduce overhead, peers can use fewer input blocks to generate a coded block [6], [7], [10], [16]. Xu et al. proposed Swifter [7], which divides a file into fixed-size generations and uses network coding at the generation level, and distributes generations based on the local-rarest-first algorithm. Ma et al. [6] designed P2P system based on sparse linear network coding, and implemented the system and measured the performance. They measured the performance of their system on PlanetLab, and compared it with their implementation of a BitTorrent-like P2P system. Their experiment found that their scheme outperforms a BitTorrent-like system both in efficiency and robustness. In comparison with their work, we use a more efficient network coding scheme, and study the performance of our scheme in PlanetLab experiments with more nodes and larger file sizes.

VI. CONCLUSION

Network coding is an emerging solution to address problems in P2P content distribution. However, There has been some doubt about the performance gains from network coding in practice. Network coding has also been blamed for its computational complexity and excessive resource use. In this paper, we answered the question: *how much does network coding benefit and cost real-world applications?*

We first sought a practical encoding scheme. According to the number of blocks to be combined for encoding, there are tradeoffs between encoding overhead and linear dependency.

There is no existing encoding scheme that satisfies both low encoding overhead and low level of block dependency. Therefore we proposed *i-Code* which is lightweight and efficient. *i-Code* combines only two blocks for every encoding operation, dramatically reducing encoding overhead. However, it does not have the dependent block penalty faced by encoding schemes which combine few input blocks.

To measure the performance and overhead in a real-world application, we implemented *i-Code* into BitTorrent clients. With these clients, we provided a thorough empirical comparison between a non-coding system (BT), and a network coding-enabled system (NC). Network coding could provide near-optimal block scheduling and provided reliable performance even when nodes behaved dynamically and unexpectedly. With measurement results, we concluded that network coding can provide improved performance and robustness with affordable overhead.

In order to put network coding into practical use, there should be a way to protect it from pollution attacks, where an attacker injects a false corrupt packet. There are some homomorphic hash functions and homomorphic signature schemes designed to deal with this problem, but no schemes have been implemented into systems and tested in real-world environments. We plan to study the performance and overhead introduced by defenses against the pollution attacks in practice.

REFERENCES

- [1] BitTorrent, Inc., “BitTorrent,” <http://www.bittorrent.com/>.
- [2] P. Rodriguez and E. W. Biersack, “Dynamic parallel access to replicated content in the internet,” *IEEE/ACM Transactions on Networking*, vol. 10, pp. 455–465, 2002.
- [3] C. Gkantsidis and P. R. Rodriguez, “Network coding for large scale content distribution,” in *Proceedings of INFOCOM 2005*. IEEE, 2005, pp. 2235–2245.
- [4] M. Izal, G. Uroy-Keller, E. Biersack, P. A. Felber, A. A. Hamra, and L. Garces-Erice, “Dissecting bittorrent: Five months in torrent’s lifetime,” in *Passive and Active Network Measurement*, 2004.
- [5] G. Ma, Y. Xu, M. Lin, and Y. Xuan, “A content distribution system based on sparse linear network coding,” in *Proceedings of the 3rd Workshop on Network Coding, Theory, and Applications (NETCOD)*, 2007.
- [6] G. Ma, Y. Xu, K. Ou, and W. Luo, “How can network coding help P2P content distribution?” in *IEEE International Conference on Communications (ICC '09)*. IEEE, 2009, pp. 1–5.
- [7] J. Xu, J. Zhao, X. Wang, and X. Xue, “Swifter: Chunked network coding for peer-to-peer content distribution,” in *IEEE International Conference on Communications (ICC '08)*. IEEE, 2008, pp. 5603–5608.
- [8] M. Yang and Y. Yang, “Peer-to-peer file sharing based on network coding,” in *Proceedings of the 28th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, 2008, pp. 168–175.
- [9] D. Bickson and R. Borer, “The BitCod client: A BitTorrent clone using network coding,” in *Peer-to-Peer Computing*, M. Hauswirth, A. Wierzbicki, K. Wehrle, A. Montessor, and N. Shahmehri, Eds. IEEE Computer Society, 2007, pp. 231–232.
- [10] C. Gkantsidis, J. Miller, and P. Rodriguez, “Comprehensive view of a live network coding P2P system,” in *Internet Measurement Conference*, 2006, pp. 177–188.
- [11] M. Wang and B. Li, “How practical is network coding?” in *Proceedings of the 14th IEEE International Workshop on Quality of Service (IWQoS)*, 2006, pp. 274–278.
- [12] —, “Lava: A reality check of network coding in peer-to-peer live streaming,” in *Proceedings of INFOCOM 2007*, 2007.
- [13] T. Ho, R. Koetter, M. Médard, D. R. Karger, and M. Effros, “The benefits of coding over routing in a randomized setting,” in *Proceedings of IEEE International Symposium on Information Theory (ISIT 2003)*, 2003, p. 442.
- [14] Holmes, D., “Enhanced CTorrent,” <http://www.rahul.net/dholmes/ctorrent/>.
- [15] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, “PlanetLab: An overlay testbed for broad-coverage services,” *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 3–12, 2003.
- [16] P. A. Chou, Y. Wu, and K. Jain, “Practical network coding,” *Allerton Conference on Communication, Control, and Computing*, vol. 41, no. 1, pp. 40–49, 2003.
- [17] D. M. Chiu, R. W. Yeung, J. Huang, and B. Fan, “Can network coding help in P2P networks?” in *Proceedings of the 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2006, pp. 1–5.
- [18] H. Schulze and K. Mochalski, “Internet Study 2008/2009,” <http://portal.ipoque.com/downloads/index/study>.
- [19] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, “Network information flow,” *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 7 2000.
- [20] S.-Y. R. Li, R. W. Yeung, and N. Cai, “Linear network coding,” *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.
- [21] R. W. Yeung, “Avalanche: A network coding analysis,” *Communications in Information and Systems*, vol. 7, no. 4, pp. 353–358, 2007.
- [22] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, “Do incentives build robustness in bittorrent,” in *Proceedings of 4th USENIX Symposium on Networked Systems Design and Implementation*, 2007.
- [23] Vuze Wiki, “Torrent Piece Size,” http://wiki.vuze.com/w/Torrent_Piece_Size.
- [24] BitTorrent, Inc., “Mainline,” <http://www.bittorrent.com/>.
- [25] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, “A digital fountain approach to reliable distribution of bulk data,” *SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 56–67, 1998.
- [26] P. Maymounkov and D. Mazieres, “Rateless codes and big downloads,” in *IPTPS*, 2003.