Paircoding: Improving File Sharing Using Sparse Network Codes

Christian Ortolf Christian Schindelhauer Arne Vater

Computer Networks and Telematics
Georges-Köhler-Allee 51
University of Freiburg, Germany
{*ortolf, schindel, vater*}*@informatik.uni-freiburg.de*

Abstract

BitTorrent and Practical Network Coding are efficient methods for sharing files in a peer-to-peer network. Both face the problem to distribute a given file using peers with different and dynamic bandwidth and only temporal availability. For this, BitTorrent partitions the files and uses the upload and download of each peer. In addition to this, Practical Network Coding uses a random linear combination of the parts. The original file can be decoded by a matrix operation as soon as enough linear combinations have been gathered at a peer.

It is known that Practical Network Coding optimizes the network flow in any peer-to-peer network, yet suffers from the cost of read/write disk operations for encoding and decoding. In this respect, BitTorrent is very efficient, yet falls behind because it has to face the coupon collector problem when distributing parts.

We present Paircoding as an alternative which is regarding filesharing at least as good as BitTorrent and shares nearly the same computational disk access complexity with BitTorrent. In some scenarios Paircoding outperforms BitTorrent regarding network flow and performs as well as Practical Network Coding. Paircoding distributes only a linear combination of two parts which alleviates the coupon collector problem of BitTorrent without the computational overhead of Practical Network Coding.

For analytical proofs of these statements we formalize filesharing in a peer-to-peer network in a round model and introduce a computational model which allows to compare the efficiency of the filesharing algorithms in a distributed environment. Since BitTorrent tries to overcome the coupon collector problem with various policies we face a family of BitTorrent systems. We show that for each BitTorrent policy there is a Paircoding policy which is at least as good regarding filesharing quality.

1. Introduction

The distribution of information to a set of hosts in a computer network is called multicasting. For the Internet IP Multicast has been provided using message duplication at router nodes [1], [2]. While this is clearly the best approach, the construction of an optimal distribution tree is NP-hard. Furthermore there is no protocol in the transportation layer implementing reliable delivery, and most Internet service providers do not offer IP multicast message service and discard such packets. As a consequence, multicasting must be implemented using point to point connections based on TCP or UDP messages. A client-server architecture can solve this problem by storing the information on the server and allowing the clients to download it. The drawback is a natural bottleneck on the server side.

Decentralized peer-to-peer communication can avoid such bottlenecks. After the introduction of peer-to-peer networks with efficient lookup service, e.g. CAN [3], Chord [4], Pastry [5], Tapestry [6], some of these networks were used to implement multicast protocols like Bayeux [7], CAN-Multicast [8], and Scribe [9]. For this, they establish multicast trees on the overlay networks. It turns out that a tree for data distribution is unfair in favor of leaf nodes, since they simply benefit by downloading the data, while inner nodes upload more data than they download. An original solution of this problem was presented with Splitstream [10] where the given file is partitioned into smaller parts and several trees are overlaid such that the upload and download of peers is balanced. However, this scheme relied on a centralized planning instance for the delivery.

1.1. BitTorrent

Bram Cohen improved this idea to a fully distributed system, called BitTorrent [11]. Delivery trees are constructed implicitly by peers interested to share (download or upload) a file. Such a file is partitioned and peers offer to upload parts as soon as they have downloaded them. BitTorrent uses incentives to discourage bad distribution behavior, called leeching when a peer downloads more than it uploads. Then the peer is banned (a.k.a. choked) from downloading further parts. This design works so well, that the peer-to-peer network users made BitTorrent the first choice to share files such that BitTorrent is dominating the global peer-to-peer network traffic for years.

A lot of research has been devoted to the question how to

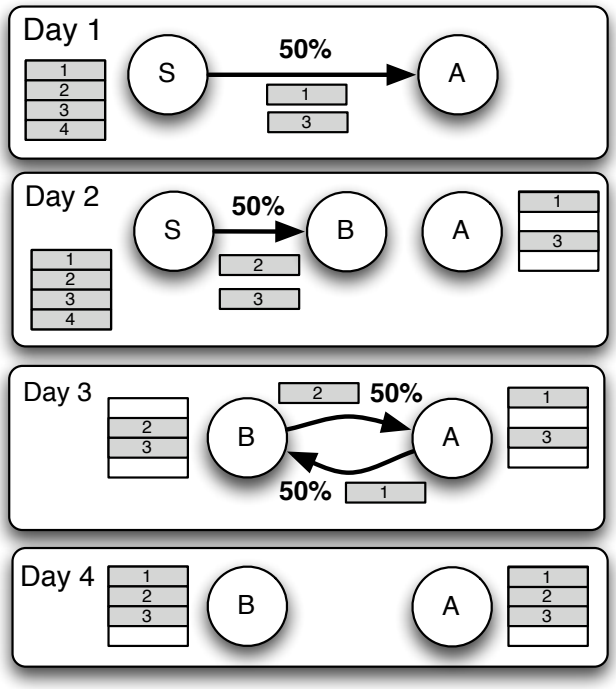


Figure 1. BitTorrent under the random selection policy

optimize such incentives using methods from game theory, e.g. [12], [13]. Different delivery strategies, called policies, lead to different behaviors. In this paper we concentrate on efficient network throughput of policies and not on fairness.

Regarding the efficiency of the network throughput BitTorrent reaches fundamental limits as the following example shows. Peer S wants to distribute a file. In BitTorrent this file X is partitioned into smaller parts, called blocks $x_1, \ldots, x_n$. On the first day only peer A is interested in this file. It downloads random parts of this file. Yet, because of a slow Internet connection it manages only to download 50% of all parts before A is disconnected from the Internet. On the next day only peer B and peer S are online and analogously B manages to download 50% of all data parts before now S is disconnected. On the last day S has disconnected and peer A and B are active. They now exchange their parts of the file and in the best case A possesses the parts that B is missing and vice versa. Then, both peers can complete the distribution process. This can be easily provided by a policy that distributes each part once. However, if the seeder S gave A and B an independent random set of blocks then A and B can only reach 75% of all parts in the expectation. One can easily extend this example with a peer C such that any policy will fail to distribute the information in the final round as we show later in Theorem 4.4.

1.2. Network Coding

No policy of BitTorrent can overcome the fundamental problem that at some time a decision has to be made which block should be distributed. Fortunately, there is an optimal solution to this problem as been introduced by the landmark paper presenting Network Coding [14]. Essentially it says, that if we present the downloading process as a network flow, then Network Coding can deliver the full file if there is a flow from the seeder to the peer of the size of the file. Furthermore, there exists a Practical Network Coding scheme [15] where network codes are simple linear combinations of blocks over a finite field.

Referring to the above example, in such a scheme the seeder would send $\frac{n}{2}$ code blocks $b_i = \sum_{j=1}^{n} a_{ij} x_j$ to peers A and B. In the last round A and B could exchange these blocks and recalculate the original code if the matrix $(a_{ij})_{i,j}$ can be inverted. A condition which can be easily guaranteed, e.g. with constant probability for a random choice of a_{ij}.

While Network Coding allows optimal usage of the network connections there is a price to pay. First, the coefficients must be known in advance or be delivered with the messages. This is only a marginal problem when the code block size is chosen large enough. Secondly, for the decoding a matrix operation has to be computed. Again a minor problem since modern computer power. And thirdly, for reconstructing the original data at a peer, a matrix vector multiplication has to be performed (using the inverted matrix). This is a major problem for large file sizes since they reside on the hard disks. So, each file must be scanned and combined just for the reconstruction of one data block. This constitutes an overhead of n read/write operations for Network Coding where n is the number of data blocks. An empirical analysis of the overhead can be found in [16] where the authors conclude that there is no coding advantage.

We think that this is the reason why network coding peer-to-peer systems like Avalanche [17] are not as successful as BitTorrent. The additional overhead after downloading the code blocks is simply not acceptable for most users: When one downloads 4 GByte of data consisting of 1000 blocks, one does not expect to perform disk operations reading 4000 GByte on each participating host while BitTorrent only writes 4 GByte of data.

1.3. Our Contribution: Paircoding

Paircoding is a sparse form of Network Coding. Here, each code block is a linear combination of two original blocks. In some situations Paircoding provides the same behavior as Network Coding while the inverse matrix can be computed in linear time and there is only a linear number of read and write accesses for decoding necessary. Like in

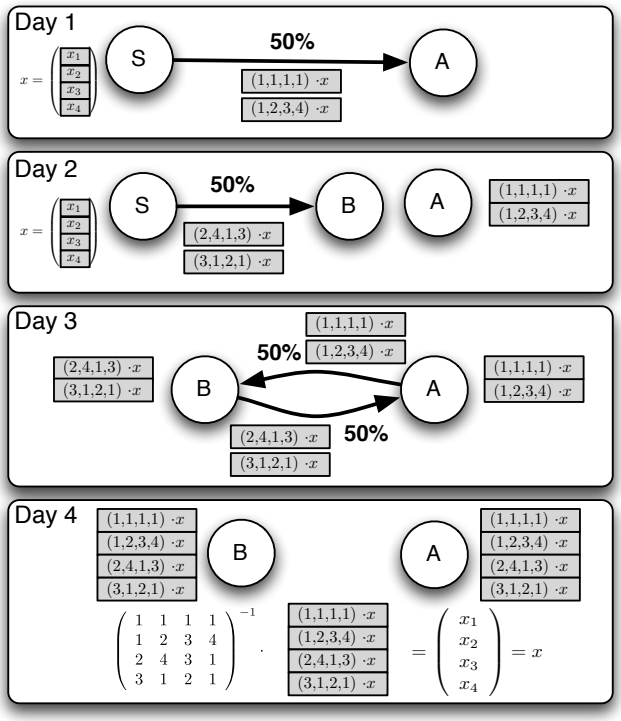


Figure 2. Network Coding allows optimal file sharing

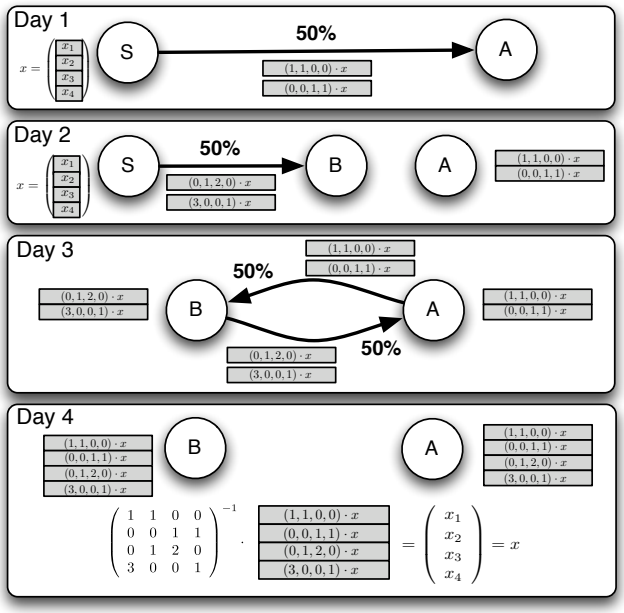


Figure 3. Paircoding is a sparse Network Coding

Network Coding, Paircoding peers can compute new code blocks from available code blocks, but not to the full extent.

2. Model Description

Throughout this paper we concentrate on distributing a single file of length m over an alphabet Σ (e.g. binary alphabet, bytes, words) which will be partitioned into n equal units of size $s = \frac{m}{n}$. We denote the blocks of the file by $x_1, \ldots, x_n$. To reconstruct the file from the blocks the order must be known. We assume that this small amount of information is implicitly provided by the communication protocols or can be delivered without extra cost.

Practical Network Coding and Paircoding use linear encodings of blocks in finite fields.

Definition 2.1 (Linear code block) *A linear code block*

$$b_c = (c_1, \ldots, c_n) \cdot (x_1, \ldots, x_n)^T$$

is a linear combination of all n blocks where c_i is an element of the finite field and x_i denotes a vector of the block size s over the finite field.

To denote a Paircoding we use the notation

$$b_{i,j,c_i,c_j} = c_i x_i + c_j x_j \ .$$

When the particular (non-zero random) choice of c_i and c_j is unimportant we write $b_{i,j}$.

Note that simple blocks can be seen as a special linear encoding with a single positive entry in the encoding vector c. In our model $P = \{p_1, \ldots, p_k\}$ is the set of peers which are interested in sharing the file. We consider a round model where the complete information of the system can be described by the **file sharing state** $\gamma(p, t)$ of each peer after round t. It is defined as the set of all code blocks that are available at peer p after round t. A seeder p is a peer with file sharing state $\gamma(p, 0) = \{x_1, \ldots, x_n\}$.

Definition 2.2 (Progress) *For a file sharing state $\gamma(p, t) = \{b_{c_1} \ldots, b_{c_r}\}$ the peer's p progress of the file is the number of linear independent encodings divided by n, i.e. $\frac{1}{n}\text{rank}({c_1}^T, \ldots, {c_r}^T)$.*

The sum of the progresses of a set of peers does not correctly describe the ability to reconstruct the whole file from the available information. To describe the reconstruction ability of several peers that may exchange an arbitrary amount of data, we use the term "availability".

Definition 2.3 (Availability) *The file availability of a set of peers is the relative number of the linear independent encodings of the union of the peers' file sharing states, i.e. the rank of the combined encoding matrix divided by n.*

If the availability is less than one, at least one block of the file is currently not available and decodable, so that the file remains incomplete even if all peers may exchange an arbitrary amount of data/blocks. A lower bound for the availability is given by the maximum progress of all considered peers.

Before the first round we face a start situation: All peers in the set of seeders possess all blocks of the file and all other peers do not have any blocks.

In each round round a subset of the peers in P is active. Each peer has a maximum download and a maximum upload transmission bound which is assigned at the beginning of each round. All active peers can send in each round linear combinations of blocks of the file sharing state of the previous rounds, where the number is limited by the upload transmission bound. Vice versa the number of received blocks must not exceed the download transmission bound. The combination of the set of active peers in a round and their transmission bounds is called the **network configuration** of this round.

At the beginning of a round the set of peers, all transmission bounds, and the encodings of all file sharing states are known to all active peers. However, it is not known which peers will participate in the upcoming rounds and how the transmission bounds will change.

Definition 2.4 (Policy) *The policy of a file sharing state is the choice of all peers which blocks will be generated and transmitted in the current round based on the available knowledge described above.*

3. Paircoding

As mentioned in the introduction, full Network Coding provides optimal results in terms of minimizing the number of downloaded blocks that are required to provide the whole original information/file. For practical utilization, however, decoding the original file from n blocks is too costly due to the necessary $O(n^2)$ read operations. Instead of full Network Coding, our Paircoding system involves only some coding, such that decoding remains reasonably cheap.

3.1. Scenario Model

Our main goal is to increase the availability of a file being shared. However, in best-case-scenarios, a simple system (e.g. BitTorrent) can be optimal.

Example 3.1 *Suppose a file is shared (seeded) by one seed and there are k leeches downloading that file. Let the file consist of a large number of blocks ($n \gg k$). Furthermore suppose the seed has a slow Internet connection, while all leeches have a very fast one.*

In this setting the time to distribute the file to all leeches is obviously lower-bounded by the time it takes the seed to upload every block once. Assuming that all leeches redistribute each block immediately, the total time to distribute all blocks (i.e. the whole file) to everyone is only little longer.

Not even full Network Coding (ignoring the decoding's complexity) could possibly improve this.

Thus, we focus on scenarios where seeds and leeches join and leave during file distribution. Especially, the seed will leave before the first peer completes the download. We model this as a round-based system. In each round blocks are exchanged between active peers that all have equal upload/download capacities. Peers may only join or leave at the beginning of a round. To model arbitrary residence times for peers, the length of rounds can be chosen small. If a peer tries to, but fails to exchange any blocks in one round, it is considered inactive. This model suffices to describe any block-based file sharing system.

3.2. Encoding

We base our system on the idea of Network Coding, but create linear combinations not from all original n blocks. Instead, the linear combinations (referred to as linear code blocks) contain only a small number of blocks, more precisely two, cp. Definition 2.1. So we use encoded pairs of plain blocks, which results in the name Paircoding. The linear coefficients c_i, c_j are non-zero and randomly chosen in finite fields and included as parameters when linear code blocks are exchanged during download. Furthermore, if possible, they are guaranteed to be linear independent. Simply put, a code block $b_{i,j}$ contains half the information of the blocks x_i and x_j it is created from. Thus two different code blocks made of the same original block contain all information (i.e. data) of that original block (if given linear independent coefficients).

3.3. Block Upload Selection

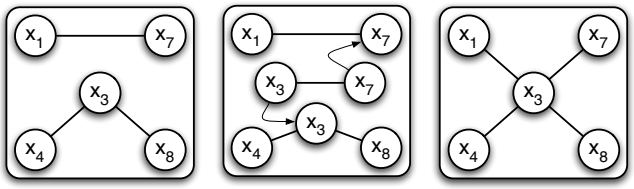
A wide range of different policies to select "good" blocks for upload is possible. Since this is a research area of its own, e.g. game theory, and lots of groundwork has been done in the context of BitTorrent, we do not focus on these aspects. However, in Theorem 4.3 we prove that Paircoding is at least as good as BitTorrent.

Encoded blocks can be created from any two available blocks. Moreover, if two code blocks are part in the same connected block component (see Definition 3.3) it is possible to create a new encoded block using any combination of two of the plain blocks involved. This works similarly to the recoding (path compression) described in the next section.

3.4. Decoding

When downloading a block $b_{i,j}$ has finished, it is added to a special data structure to ease decoding. In the following, we describe this data structure and how it is created while downloading.

Definition 3.2 (Connected code blocks) *We call two code blocks $b_{i,j}$ and $b_{i',j'}$ connected, if they are both created of at*



an encoded block $b_{1,7}$ and two connected blocks below

adding the new block $b_{3,7}$

the star component after recoding $b_{1,7}$ with x_3 as the head block

Figure 4. Connected block component

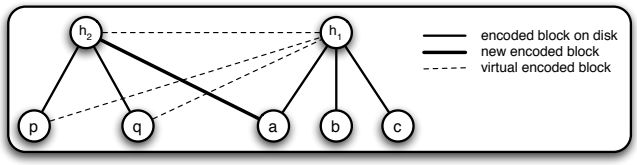


Figure 5. Merging two star components

least one same block, i.e. if exits $z \in \{i, j\}$ *and* $z' \in \{i', j'\}$ *such that* $z = z'$.

Note that two blocks may be single-connected if they share only one block, or double-connected if they are both created from the same two blocks. However, in that case the linear coefficients must be independent, otherwise they would – except for a factor – be exactly the same code blocks, together yielding no more information than only one of them.

For visualization we draw an encoded block $b_{i,j}$ as a graph with two nodes representing the original blocks from $\{x_1, \ldots, x_n\}$ and an edge between them, indicating the code block created from them. If two blocks are single-connected, they are represented by a graph with three nodes and two connecting edges, see Figure 4. The block represented by the node with the highest degree is then called the **head block**. If more than two code blocks are connected, they can be recoded such that one block is the head and all code blocks consist of this head and another block.

Definition 3.3 (Connected block component) *A connected block component is a set of* ℓ *linear independent connected blocks in which all encoded blocks* $b_{i,j}$ *can be recoded to* $b_{h,i}$ *or* $b_{h,j}$ *if* $i, j \neq h$*, and* h *is the head block. Its number of connected blocks* ℓ *is called its size.*

If the recoding is finished for all blocks, we call the block component a star component with head h.

For an example see Figure 4, on the right. The head block is x_3 and the encoded blocks are $b_{3,1}$, $b_{3,4}$, $b_{3,7}$, and $b_{3,8}$.

The coding matrix of a connected block component of size ℓ has the form $(n \times \ell)$, and its rank is $\ell - 1$. After downloading $b_{i,j}$ one of the following actions are applied:

1) If neither block i nor j is present in any connected block component, a new one is created, with i as its head.
2) If w.l.o.g. only the block i is present in a star component with head h, the new block is recoded to $b_{h,j}$ and added to the component. This is done by using the present block $b_{h,i}$ with its linear coefficients c_h, c_i and the new block $b_{i,j}$ with its linear coefficients $c_{i'}, c_j$ to calculate $b_{h,j} = c_{i'}^{-1} c_j \cdot b_{i,j} + b_{h,i}$. The actual coding is deferred until the block is read, though.
3) If both blocks i and j are present in two different star components, they become connected by the new block. Then all connected code blocks from the smaller star component are recoded and added to the bigger one, effectively merging both stars into a larger one. The actual coding operations are also deferred until the encoded block is read. Thus, the star component becomes equivalent to a disjoint-set forest structure [18], where our (deferred) coding operation is equivalent to the path compression, see Figure 5. Note that reading a block will occur not only at the final decoding of a star component, but also on uploading to another peer, thus also initiating a path compression.
4) If both blocks are present in the same star component it is added to it. If the new linear combination of $b_{i,j}$ is linear independent of all other blocks in the component, the rank of the coding matrix $(n \times \ell)$ increases by one to ℓ. Hence all blocks of the component can be decoded.

Lazy coding. To optimize the computation and read/write overhead, we apply a "lazy" coding scheme similar to path compression in a disjoint-set forest structure. Thus, any coding operation is delayed and only a virtual code block is created to indicate that a particular coding is possible. This coding is performed upon request, i.e. when a virtual code block has to be read. The component given in Figure 5 has three virtual code blocks b_{h_1,h_2}, $b_{h_1,p}$, and $b_{h_1,q}$. If for example $b_{h_1,q}$ is requested for reading, it will be created, effectively removing $b_{h_2,q}$. Thus, at some time, the connected block component will be recoded to a star component.

Decoding a star component can be done fast by decoding the head block first. After that, decoding all other blocks is simple, since they are all connected to the head, see Figure 6.

4. Analysis

We now present an analytical method to compare the effectiveness of file sharing systems and compare BitTorrent, Network Coding, and Paircoding.

Definition 4.1 (Performance relation) *A file sharing system* S_1 *is at least as good as a system* S_2*, noted as*

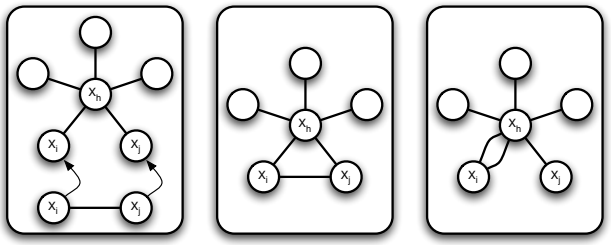


Figure 6. Decoding star components

$S_1 \geq S_2$, if both systems begin with the same starting situation and are confronted with the same sequence of network configurations. Then, for every peer the progress in S_1 is at least as large as its progress in S_2.

If there is a starting situation and a sequence of network configurations where a peer in S_1 has larger progress than this peer in S_2 and if $S_1 \geq S_2$, we say S_1 outperforms S_2, i.e. $S_1 > S_2$.

Corollary 4.2 *To distribute file $X = (x_1, \ldots, x_n)$, Network Coding requires only the minimal number of blocks n, which is optimal.*

Theorem 4.3 *Paircoding shares files at least as good as BitTorrent:*

$$\text{PAIRCODING} \geq \text{BITTORRENT}$$

Proof: Let the peers $P = p_1, \ldots, p_k$ distribute file $X = (x_1, \ldots, x_n)$ and $\mathcal{P}_{\text{BT}}$ an arbitrary BitTorrent policy. Then, according to $\mathcal{P}_{\text{BT}}$ a peer p with file sharing state $\widetilde{\gamma} = \gamma(p, t)$ selects the block

$$\mathcal{P}_{\text{BT}}(\widetilde{\gamma}) = x_i$$

for distribution, $1 \leq i \leq n$. Then, when using Paircoding and its policy $\mathcal{P}_{\text{PC}}$, p chooses

$$\mathcal{P}_{\text{PC}}(\widetilde{\gamma}) = b_{x_i, x_{n-i}}$$

Thus, when every block x_i has been selected once for distribution by $\mathcal{P}_{\text{BT}}$, i.e. the whole file is distributed, it has been selected for encoding in a Paircoding block exactly twice by $\mathcal{P}_{\text{PC}}$. This is sufficient to allow the decoding of every block and hence delivers the original file with the same amount of distributed data as required by BitTorrent. □

Theorem 4.4 *For some scenarios, Paircoding performs as good as Network Coding and better than BitTorrent.*

Proof: We describe a scenario that consists of four discrete rounds, all of the same length. We assume that upload and download capacities of all peers are equal. As the length of one round we choose the time it takes to transmit half of file X. First we will analyze the performance of Paircoding and after that we will point out the differences

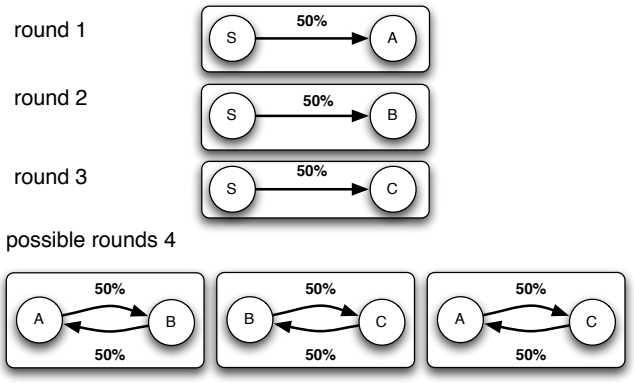


Figure 7. In this scenario Paircoding outperforms BitTorrent

of BitTorrent. Finally we apply the scenario to Network Coding.

We consider the following round-based scenario with one seed and three leeches. The seed p_1 provides the complete file $X = (x_1, \ldots, x_n)$, $n \in \mathbb{N}$ and n even, desired by all three leeches p_2, p_3, and p_4. At the beginning the progress of $p_2, \ldots, p_4$ is zero. In the first three rounds, the seed and exactly one of the leeches are active. During one round, the seed can upload 50% of the file size, i.e. $\frac{n}{2}$ (encoded) blocks can be downloaded by the active leech in one round, cp. Figure 7.

Paircoding. In rounds $r \leq 3$ the seed creates and transmits the encoded blocks

$$b_{1,2,c_1^r,c_2^r}, b_{3,4,c_3^r,c_4^r}, \ldots, b_{n-1,n,c_{n-1}^r,c_n^r}$$

In each round the linear coefficient used for block $b_{i,j}$ must be linear independent from the ones used in previous rounds:

$$\forall \delta, r_1, r_2 \mid r_1 \neq r_2 : (c_i^{r_1}, c_j^{r_1})^T + \delta \cdot (c_i^{r_2}, c_j^{r_2})^T \neq 0$$

In the final round $r = 4$, any two of the three leeches $p_2, \ldots, p_4$ are active, while the third one and the seed are not available. W.l.o.g. let p_2 and p_3 be active. Again, both can download $\frac{n}{2}$ encoded blocks (now simultaneously while uploading). Thus, all data between p_2 and p_3 is exchanged. Since then two encoded blocks $b_{i,j,c_i^{r_1},c_j^{r_1}}$ and $b_{i,j,c_i^{r_2},c_j^{r_2}}$ from the rounds r_1, r_2 are available to each of the active peers, and the linear coefficients $(c_i^{r_1}, c_j^{r_1})^T$ and $(c_i^{r_2}, c_j^{r_2})^T$ are linear independent, all blocks can be easily decoded by solving

$$\begin{pmatrix} x_i \\ x_j \end{pmatrix} = \begin{pmatrix} b_{i,j,c_i^{r_1},c_j^{r_1}} \\ b_{i,j,c_i^{r_2},c_j^{r_2}} \end{pmatrix} \cdot \begin{pmatrix} c_i^{r_1} & c_j^{r_1} \\ c_i^{r_2} & c_j^{r_2} \end{pmatrix}^{-1}$$

Hence, for any possible last round, every original block can be decoded, yielding a progress of one, i.e. the whole file X, for both active peers. Consequentially, the availability of X also equals one.

Note that in this scenario the seed is not necessarily required to use the same block combinations for encoding in each round. However, it is important that in each round each block $x_1, \ldots, x_n$ is used exactly once in an encoded block.

BitTorrent. We show that for our scenario the upper bound of availability that BitTorrent can guarantee after the final round is $\frac{5}{6}$ and hence less than achieved by Paircoding.

We model the scenario as a two-player-game. The first player may choose the blocks in each round with full knowledge of rounds one to three. He tries to maximize the availability in round four, while the secondary player acts as adversary who chooses the two peers in round four to minimize the availability. The following strategies hold for player one:

- Send each block at least once. Otherwise, one block is completely missing for any peer combination in round four.
- No peer receives a block more than once.
- No block is sent more than twice, since two copies of a block suffice to guarantee its presence in round four.
- Minimize the overlapping of blocks between two peers for any combination of them.

The adversary merely selects the combination with the lowest availability in the last round. This is equivalent to select the combination with the largest overlapping of blocks, which is the reason for the last rule of player one.

Let A, B, C the set of blocks at peers p_2, p_3, p_4, and each peer receives one half of the whole file, i.e. $|A| = |B| = |C| = \frac{1}{2}n$. W.l.o.g., let $|A \cap B| = x$ and $|A \cap C| = y$. Then it follows

$$|B \cap C| = z = \frac{1}{2}n - x - y$$

For symmetry reasons $x = y = z$, as any asymmetry helps the adversary. So,

$$x = \frac{1}{2}n - 2x = \frac{1}{6}n$$

The overlap between any two peers is $\frac{1}{6}n$. This leaves the relative information available in round four of $\frac{n-\frac{1}{6}n}{n} = \frac{5}{6}$ which proves the claim.

Network Coding. For Network Coding, any n encoded blocks are sufficient to decode X, if the linear coefficients are pairwise linear independent. Under this assumption, any two peers can decode X in the last round after exchanging their data. This yields the same performance for both Paircoding and Network Coding in this scenario. Furthermore, Network Coding has the same values for availability in each round.

This concludes the proof of the theorem. □

We assumed that upload and download do not interfere and all data can be exchanged in the last round. Instead of this assumption, we could also double the duration of the last round, or even increase it to infinity. The results would remain the same.

Corollary 4.5 *Paircoding outperforms BitTorrent, i.e.*

$$\text{PAIRCODING} > \text{BITTORRENT}.$$

5. Cost of Paircoding

Definition 5.1 (Read/write cost) *We define read and write cost as the total number of blocks that need to be read or written until the file is decoded and saved on hard disk and denote it by C_r and C_w, or C_{rw} respectively.*

Since reading and writing could have different delays on hard disks, we may distinguish between read and write cost. In our case, we consider blocks as information pieces. File sharing in general obviously requires at least one read operation at the sender and one write operation at the receiver, so $C_r = C_w = 1$ is a general lower bound.

Theorem 5.2

1) *Paircoding has worst case read/write cost of $O(n \cdot \alpha(n, n))$, where $\alpha(x, y) = \min\{i \geq 1 : A(i, \lfloor x/y \rfloor) > \log y\}$ is the inverse of Ackermann's function.*
2) *Paircoding requires disk space m if m is the size of the shared file.*

Proof:

1) We assume that any computation can be done in main memory and does not require additional disk access. I.e., the block size should be small enough to easily fit into main memory. However, during download, blocks are saved on disk in order not to loose any data in case the download is interrupted.
   The read cost for the seed are obviously $O(n)$, since at most two blocks must be read to create one encoded block for upload. Depending on the upload policy and implementation optimization, the constant factor may be as small as $(1 + \epsilon)$, $\epsilon > 0$. This can be achieved when consecutively sending the encoded blocks $b_{1,2}, b_{2,3}, \ldots, b_{n-1,n}, b_{n,1}$. Here, the second original block of an encoded block is used as the first original block in the next encoded block, thus it does not have to be read again.
   As described in Section 3.4, the read/write cost for peers correspond to the computation cost of union find with path compression for disjoint-set forrest structures. This is known to be worst-case bounded by $O(n \cdot \alpha(n, n))$, which prooves the claim. □
2) Since the whole coding process can be done in-place and only requires additional main memory (cp. Section 3.4), no additional disk space is required. □

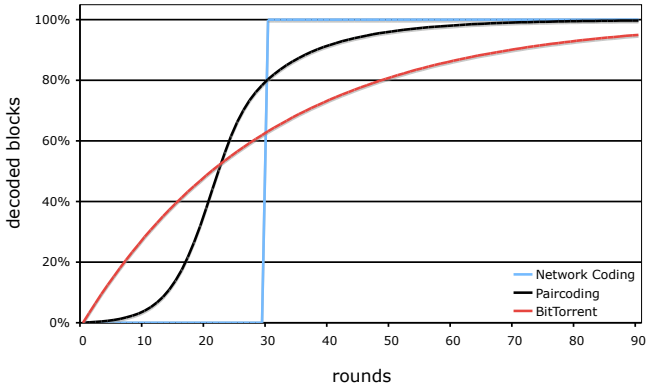


Figure 8. Simulation of decodability for one peer

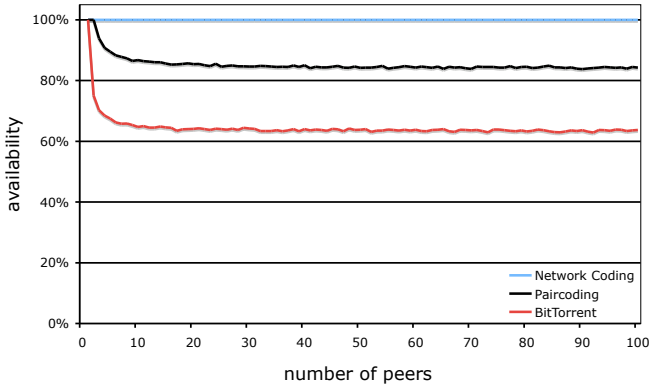


Figure 9. Simulation of availabilty for increasing number of peers

6. Empirical Tests

To compare Paircoding, Network Coding, and BitTorrent regarding the coupon collector problem, we simulated the following scenario. One seed distributes a file of $n = 30$ blocks to a leech under the uniformly distributed random selection policy. For BitTorrent the seed randomly selects one block for upload, while for Paircoding the seed randomly chooses two blocks and then uploads a linear combination (code block) of them. In case of Network Coding each code block is a linear combination of all n blocks. The results, averaged from 100,000 simulation runs, are shown in Figure 8. The curves denote the percentage of decoded (available) blocks at the receiver peer.

While the amount of blocks transferred is identical in each round, the numbers of decodable blocks are very different: Until round 22, BitTorrent can provide more data because Paircoding cannot decode the received code blocks, yet. After that, however, Paircoding overtakes and has clearly more success in providing new data. Here, we have considered the situation from the user's perspective. In fact, the progress of Paircoding outperforms BitTorrent from the start, since the probability of transmitting linear dependent pair codes is much smaller than the probability of transmitting the same block. Network Coding requires exactly n independent code blocks to decode everything.

We also simulated the performance regarding the availability of a file with $n = 100$ blocks depending on the number p of downloading peers. In this setting, each peer receives $\lfloor n/p \rfloor$ blocks from a seed. Additionally, $n - p \cdot \lfloor n/p \rfloor$ peers receive one extra block, so that the sum of all downloaded blocks equals n. For each peer the selection of blocks is coordinated, i.e. no peer receives two redundant (either identical or linear dependent) blocks. Furthermore, in case of Paircoding, no original block is selected more than once per peer for encoding, if $p \geq 2$. Coordination between peers is not allowed, though. Finally, the availability of the file is measured, if all downloading peers are active, while the seed is inactive. This setting is closely related to the scenario we used in the proof of Theorem 4.4.

The diagram in Figure 9 shows the results averaged from 100 simulations for an increasing amount of peers up to $p = 100$. All three systems start with availability one, since block selection can be coordinated for a single peer so that the sole peer receives n independent blocks. When increasing the number of peers, the curve of Paircoding converges quickly to $1 - \frac{1}{e}$, according to the coupon collector problem. In case of Paircoding and two peers, the availability is still one, cp. scenario in Figure 7. For $p \geq 3$ the curve converges only slightly slower to a value of approximately $1 - \frac{1}{e^2}$. Network Coding provides full availability in all cases.

7. Summary and Outlook

We have introduced a new sparse network coding scheme, called Paircoding, which is based on linear combinations of two file blocks and we have compared this scheme with BitTorrent and Network Coding. Paircoding requires a near optimal number of disk read/write operations while providing some benefits from Network Coding. We have given analytical proofs that the number of read/write operations is only a factor of $\alpha(n, n)$ higher than the optimal linear number of operations e.g. required by BitTorrent (where $\alpha(n, n)$ is the inverse Ackermann function). Other network coding schemes provide a much higher number of disk operation which increase by number of partitions of the original file.

To compare the success of a file sharing systems and the possibility to overcome the coupon collector problem we have introduced a complexity measure for block based file sharing systems. With this methodology we can show that Paircoding outperforms BitTorrent while Network Coding outperforms all of them. In some instances Paircoding can keep up with Network Coding where BitTorrent does not.

In the tradeoff situation of disk operations and file sharing effectiveness Paircoding ranks between the other schemes as an excellent compromise. These theoretical results have been backed by simulation.

References

[1] D. Waitzman, C. Partridge, and S. Deering, "Distance vector multicast routing protocol," Internet Engineering Task Force, RFC 1075, Nov. 1988. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1075.txt

[2] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, and C. Liu, "Protocol independent Multicast-Sparse mode (PIM-SM): protocol specification," Internet Engineering Task Force, RFC 2362, Jun. 1998. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2362.txt

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Computer Communication Review*, vol. 31. Dept. of Elec. Eng. and Comp. Sci., University of California, Berkeley, 2001, pp. 161–172.

[4] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01)*, ser. Computer Communication Review, R. Guerin, Ed., vol. 31, 4. New York: ACM Press, Aug. 27–31 2001, pp. 149–160.

[5] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Lecture Notes in Computer Science, In Proc. of the International Conference on Distributed Systems Platforms (IFIP/ACM),*, vol. 2218, pp. 329–350, 2001.

[6] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao, "Distributed object location in a dynamic network," in *SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM Press, 2002, pp. 41–52.

[7] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of NOSSDAV*, June 2001.

[8] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," in *Networked Group Communication, Third International COST264 Workshop, NGC 2001, London, UK, November 7-9, 2001, Proceedings*, ser. Lecture Notes in Computer Science, J. Crowcroft and M. Hofmann, Eds., vol. 2233. Springer, 2001, pp. 14–29.

[9] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications (JSAC)*, vol. 20, no. 8, pp. 1489–1499, 2002.

[10] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," 2003.

[11] B. Cohen, "Incentives build robustness in BitTorrent," Berkeley, CA, USA, 2003.

[12] D. Levin, K. Lacurts, N. Spring, and B. Bhattacharjee, "Bittorrent is an auction: analyzing and improving bittorrent's incentives," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 243–254, 2008.

[13] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent?" in *NSDI'07*, Cambridge, MA, April 2007.

[14] 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, 2000.

[15] P. Chou, Y. Wu, and K. Jain, "Practical network coding," 2003.

[16] M. Wang and B. Li, "How practical is network coding?" *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on*, pp. 274–278, June 2006.

[17] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *INFOCOM*. IEEE, 2005, pp. 2235–2245.

[18] Z. Galil and G. F. Italiano, "Data structures and algorithms for disjoint set union problems," *ACM Comput. Surv.*, vol. 23, no. 3, pp. 319–344, 1991.