Classifying Peer-to-Peer Network Coding Schemes

Christian Ortolf Christian Schindelhauer Arne Vater

Department of Computer Science
University of Freiburg
Georges-Köhler-Allee 51
Freiburg im Breisgau, Germany
{ortolf, schindel, vater}@informatik.uni-freiburg.de

ABSTRACT

Modern peer-to-peer file sharing systems distribute large files among peers using block partitioning. Blocks can be redistributed by a peer even before the whole file is available which highly decreases the distribution time. All peer-to-peer networks face the problem of dynamic participation of the peers and dynamic bandwidth in the network. A leaving peer can cause an unrecoverable loss of blocks and obstruct further downloads of the file. Furthermore, the choice which block needs to be sent to which peer is a hard question. A random choice leads to the coupon collector problem which decreases the transmission rate. Filesharing networks like BitTorrent or Splitstream face such problems.

Network Coding overcomes this problem by using error redundant codes of all blocks of the file. An efficient randomized variant of it, Practical Network Coding, transmits and recombines random linear combinations of the blocks of the partitioned file. As soon as enough linear combinations have been gathered, the original file can be decoded by a matrix operation, optimizing the network flow in any peer-to-peer network.

All known Network Coding schemes, however, suffer from a quadratic cost of read/write disk operations for both encoding and decoding. Since there is an increasing gap between the speed of mass storage devices and the main memory, this poses an obstacle to a wider use of Network Coding schemes.

In this paper we present and investigate new network coding schemes, which form a compromise between Network Coding and uncoded block transfer schemes like BitTorrent. These schemes, called Paircoding and Treecoding have smaller read/write costs for encoding and decoding than Practical Network Coding and higher throughput than BitTorrent.

We develop a new framework for comparing the throughput of data (performance) of such peer-to-peer file sharing systems and classify these systems, as well as a BitTorrent variant which uses forward error correction. The dynamics of peer-to-peer networks are described by a round model where the set of participating peers and their link quality changes after each round. The framework compares two schemes for all possible dynamic scenarios. If the transmission rate of scheme A is at least as well as scheme B, then we say A performs as well as B. If this is the case and there is a scenario where A is better than B, we say A outperforms B. We show that all of our proposed coding schemes outperform BitTorrent, while being outperformed by Network Coding.

This leads to a hierarchy, where BitTorrent is the worst performer and Network Coding is the best performer regarding throughput. Regarding computation (disk read/write) complexity for decoding, BitTorrent and Foward Error Correction have linear time behavior, for Paircoding it is almost linear, Treecoding with one coding tree needs time $O(n)$ and Network Coding has time $O(n^2)$.

SPAA'09, August 11–13, 2009, Calgary, Alberta, Canada.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems — *Distributed applications*;
E.4 [**Data**]: Coding and Information Theory — *Nonsecret encoding schemes*;
F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity — *Nonnumerical Algorithms and Problems*

General Terms

Algorithms, Performance

Keywords

Peer-to-Peer Networks, BitTorrent, Network Coding

1. INTRODUCTION

Information distribution is one of the central objectives in the Internet. For small documents, like web pages, it is possible to separately deliver it to each requesting host. For larger documents and many requesting hosts this leads to a bottleneck at the server's side. This problem can be resolved by the IP Multicast protocol where routers duplicate packets and relieve the bottleneck at the server [17, 6]. While this is clearly the best approach, IP Multicast provides only suboptimal distribution trees because of the NP-hardness of this problem. More important, the IP Multicast protocol does not provide reliable delivery, and it is not supported by most

Internet service providers at all. So, IP Multicast remained on the fringes of the Internet. Yet, high user demand lead to the implementation of multicasting on overlay networks using point-to-point TCP or UDP messages.

Most peer-to-peer networks are designed for data distribution. The hype started with Napster and Gnutella. Then, in the first years the main focus was to find efficient lookup services and robust network structures, e.g. CAN [12], Chord [16], Pastry [15], and Tapestry [8]. For some of these networks extensions were proposed for efficient multicast like Bayeux [19], CAN-Multicast [13], and Scribe [3]. They provide multicast trees, filling the gap of multicasting in the IP layer.

Multicast trees favor leaf peers, since they do not need to upload data which is generally the scarce resource due to asymmetric Internet connections. A fair solution for all peers was presented with Splitstream [2]. In Splitstream each file is partitioned into smaller parts and multiple trees are overlaid to balance both upload and download of all peers.

BitTorrent.

Inspired by the approach of Splitstream, BitTorrent [5] was designed and became the most successful block-based file sharing system. BitTorrent does not use any encoding except for error detecting hash values of each block which does not contribute to data throughput. A downloading peer can retrieve blocks from several peers in parallel. When downloading, a peer can upload already completed blocks. The decision which blocks to distribute is made ad-hoc taking the behavior of other peers into account. This mechanism is called *policy* (see Definition 4) and it determines which block should be uploaded to whom. BitTorrent uses incentives to discourage bad distribution behavior, called leeching, when a peer downloads more than it uploads. Eager uploading is encouraged by sending such peers more blocks. This game-theoretic setting maximizes network throughput.

A lot of research has been devoted to the question how to optimize such incentives using methods from game theory, e.g. [9, 11]. While this explains the speed and the popularity of BitTorrent to a great deal we concentrate in this paper not on the game-theoretic aspects which is a vivid area of research at the moment. We concentrate on unsolvable shortcomings of a block-based approach which cause delivery failures because of missing blocks. BitTorrent is not capable to compensate a missing block. Whenever there is a block, that is not available at any of all participating peers (i.e. the *availability* is less than one, see Definition 3), none of them can ever finish the download.

Network Coding.

An optimal solution to the problem of availability is provided by Network Coding. The Practical Network Coding scheme [1, 4] gives an efficient implementation. Instead of a plain block, a linear combination of all blocks is created and transmitted. Each block is interpreted as a vector over a Galois field. Then, if all linear coefficients used are linearly independent, any n code blocks are sufficient to decode the original file. Such linearly independent vectors can easily be achieved with high probability by a random choice if the order of the finite field is chosen large enough.

Furthermore, new encodings can be computed at peers which do not have enough code blocks to decipher the original file. For this a peer uses a random linear combination of all received blocks. So, a peer does not need to decide which of the already received blocks should be forwarded. It *recodes* the code dynamically.

Definition 1. If only the seed is allowed and able to create code blocks with linear coefficients, and forwarding peers can only redistribute these code blocks, we call this a **static coding scheme**. If any peer is allowed and able to create new code blocks from other code blocks already present (recoding), we refer to this as a **dynamic coding scheme**.

We will discuss coding schemes that are unable to create new code blocks without a prior decoding of the original file, e.g. Treecoding (see Section 5). There are other coding schemes, e.g. FEC (see Section 2), that could use recoding, but they may increases the decoding complexity up to that of Network Coding, which is not reasonable as it would make the scheme obsolete.

The drawback of Practical Network Coding is its high computational overhead along with its many disk access operations: To decode one block it is necessary to read all code blocks from the hard disk, leading to a disk reading complexity of $O(n^2)$ for decoding n blocks of information. Some system approaches like Avalanche [7] use Network Coding. Yet, such schemes by far did not reach the success of BitTorrent. Moreover, some empirical analysis of the overhead conclude that there is no or only little advantage in such a coding for peer-to-peer systems [18].

The Model.

We follow the model of [10] presenting Paircoding. In this section we give a brief summary of it and refer to [10] for a more detailed 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 $s = \lceil \frac{m}{n} \rceil$. We denote the blocks of the file by $\vec{x}^{\,T} = (x_1, \ldots, x_n)^T$ as vectors over a finite field. The last block may be filled up with zeros. We assume $n = 2^i, i \in \mathbb{N}^+$ since this supports the binary presentation of data. 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 that this extra cost can be neglected.

All coding schemes presented use linear encodings of blocks in finite fields:

Definition 2. A **linear code block**

$$b_{\vec{c}} = (c_1, \ldots, c_n) \cdot (x_1, \ldots, x_n)^T = \vec{c} \cdot \vec{x}^{\,T}$$

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

To denote a code block created with a vector $\vec{c}$, where $\vec{c}$ has non-zero values at positions $a_1, \ldots, a_j, j \in \{1, \ldots, n\}$, we write $b_{\vec{c}}(a_1, \ldots, a_j)$ or, if the non-zero values are not important, $b(a_1, \ldots, a_j)$.

In both notations the i's represent the position of a block in the original file order. Note that simple blocks can be seen as a special linear encoding with a single non-zero entry in

the encoding vector $\vec{c}$. For decoding the receiver must know these coefficients. Again we assume that this small amount of information (compared to the block size) is implicitly provided by the communication protocols or that this extra cost can be neglected.

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 in each round peers can upload and download certain numbers of blocks. Before the first round we face a start situation: A set of seeding peers has all blocks of the file and all other peers do not have any blocks.

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 seed P is a peer with file sharing state $\gamma(P, 0) = \{x_1, \ldots, x_n\}$.

In each 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. With this approach we model the bottleneck of data throughput that usually appears in the last mile of Internet connections caused by asymmetric low bandwidth dial-up or DSL connections. In each round all active peers can send code blocks, i.e. linear combinations of blocks of the peer's file sharing state of the previous rounds, where the outgoing number is limited by the upload transmission bound and the ingoing number of blocks is limited by 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. An example for a network configuration is given in Figure 1.

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.

To measure the performance of a coding scheme, we use the progress and the availability. While the former is more related to the performance of a single peer, the latter measures the success of all peers currently active.

Definition 3. For file sharing state $\gamma(P, t) = \{b_{c_1} \ldots, b_{c_r}\}$ the **progress** of the file at peer P is the number of linearly independent encodings available at P divided by n, i.e. $\frac{1}{n} \cdot \text{rank}({c_1}^T, \ldots, {c_r}^T)$. The file **availability** of a set of peers is the relative number of the linearly independent encodings of the union of the peers' file sharing states, i.e. the rank of the combined encoding matrix divided by n.

In other words, the availability measures the share of information fraction that is currently available, if all active peers share their information. Obviously, this value equals one if any seed is present.

Definition 4. The **policy** of all peers with a given file-sharing state is the decision, which blocks will be generated and transmitted in the current round based on the available knowledge described above.

The policy of a file sharing system has great impact on its performance. However, in this paper we do not concentrate on the optimization of distribution policies, but we analyze and present different coding schemes. Thus we either stick to general and simple policies, e.g. random selection, or our analyses remain policy-independent.

To classify different coding schemes, we use a relation that is based on the progress of the peers involved.

Definition 5. A file sharing system S_1 **performs at least as well** as a system S_2, noted as $S_1 \geq S_2$, if both systems, started from the same situation and confronted with the same sequence of network configurations, experience that every peer in S_1 has at least a progress as large as its progress in S_2. If systems S_1 and S_2 allow different policies, then S_1 performs at least as well as S_2 if for each policy of S_2 there is a policy in S_1 which performs at least as well.

If $S_1 \geq S_2$ and in addition there exists a starting situation and a sequence of network configurations where a peer in S_1 (with a given policy) has larger progress than this peer in S_2 (for all possible policies), we say S_1 **outperforms** S_2, i.e. $S_1 > S_2$.

The other important measure is the read/write cost. We mentioned above, that network coding is capable of optimally solving the problem of data distribution in terms of progress and availability. The drawback is its high computational cost, which is dominated by the read/write accesses of the hard disks. Usually, the user's main focus is on the decoding of the file. Therefore, we concentrate on these costs in this paper. We are aware that seeds and peers that need to update the coding for retransmission have additional read/write costs. Since this time takes place during the transmission which is commonly much slower it does not incommode the user.

Definition 6. Every peer uses main memory which can store a constant (non-zero) number of blocks. We define **read and write cost** as the total number of blocks that need to be read from or written to disk to decode the file and save it on hard disk at a receiving peer. We denote this cost by C_{rw}.

2. FORWARD ERROR CORRECTION

Forward error correction (FEC) as a compromise between BitTorrent and Network Coding has been proposed and evaluated in [7]. The idea is to mainly use plain blocks for distribution and add c linearly independent complete network code blocks, i.e. using κ Reed-Solomon code blocks [14]. Each such network code block can compensate the loss of one plain block. Thus, at least n plain blocks and FEC blocks are necessary and required to decode the file. We do not consider the recoding of FEC blocks, since on the long run (i.e. for an arbitrary amount of peers) this would end up in an arbitrary number of FEC blocks, even if the number of recodings allowed for each peer is limited. This would be equivalent to Network Coding. So, forward error correction is a static coding scheme and we denote it by FEC(κ). As a reasonable block selection policy one could start with downloading all κ code blocks followed by $n-\kappa$ plain blocks.

For the computation of a single FEC block all n blocks need to be combined. So, the read cost is κn and the write cost is κ. Yet, the time complexity refers only to the decoding.

LEMMA 1. *FEC(κ) has read/write cost*

$$C_{rw}(FEC(\kappa)) = O(\min\{\kappa n, n^2\})$$

PROOF. For decoding, a peer needs at most n blocks where some $\ell \leq \kappa$ blocks are FEC blocks. With $O(\ell(n-\ell))$

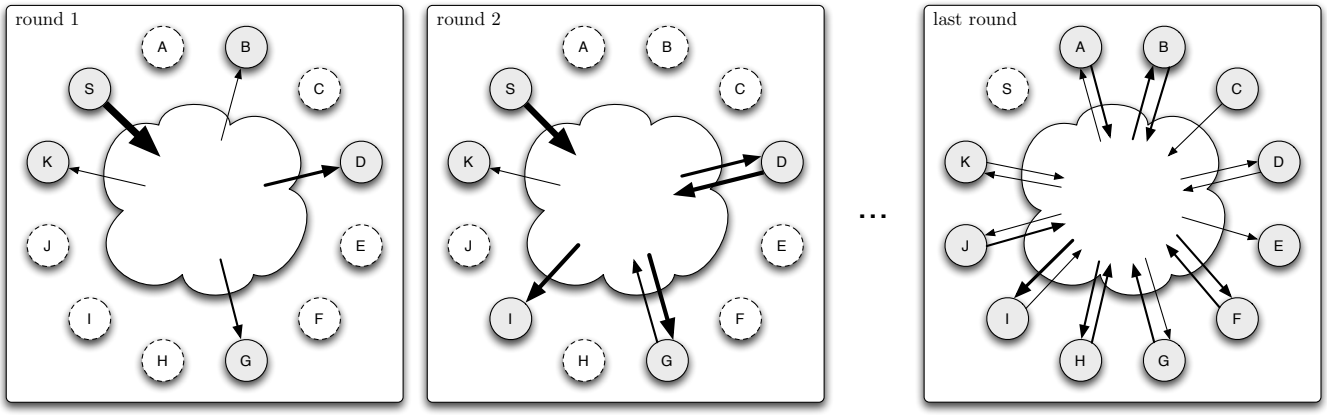

Figure 1: Example for network configurations for $P = \{A, \ldots, K, S\}$, where S is the only seed. Active peers are colored gray and their transmissions are indicated by the thickness of the ingoing and outgoing arrows. In the first round the seed is the only uploading participant. In the second round peers D and G also start to upload data, thus increasing the total network flow. The last round depicts a situation where the seed is inactive. We use this situation in some of our proofs.

additions and scalar multiplications the uncoded blocks can be removed from the linear combination of the FEC blocks and the coefficients. It remains to compute an inverse matrix of the residual $\ell \times \ell$-matrix (which we assume can be performed in main memory and thus does not add costs to the disk time measure) and multiply the resulting inverse with the vector of former FEC blocks. This adds costs of $O(\ell n)$. Summarizing we have the relevant disk time cost for decoding of $O(\ell n) = O(\kappa n)$. □

The theorem below classifies FEC as a coding class between BitTorrent and Network Coding.

THEOREM 1. *For all $\kappa \geq 0$ we have*

1. FEC(0) = BITTORRENT
2. FEC(κ) < NETWORKCODING
3. FEC(κ) < FEC($\kappa + 1$)

PROOF.

1. By definition FEC(0) and BitTorrent are equivalent.
2. Consider a policy for FEC(κ). Then we can find a policy for Network Coding such that both systems are equivalent. For this, the Network Coding policy creates the very same code blocks (or uncoded blocks) as FEC(κ) which proves FEC(κ) ≤ NETWORKCODING.

   To prove the superiority of Network Coding we consider the following scenario. In round i only seed S and peer P_i are active ($i = \{1, \ldots, \kappa + n + 1\}$) and S can transmit only one block to P_i. Finally in round $\kappa + n + 2$ the seed is inactive and n random (non seed) peers are active. This scenario is depicted in Figure 2.

   In round $\kappa + n + 1$ every policy for FEC(κ) has either transmitted a block twice or at least one of the peers P_i has no block. So, in the case of a duplicated block with probability of at least $\frac{n(n-1)}{(\kappa+n+1)^2}$ the duplicated block occurs twice in the last round and therefore the availability is at most $1 - \frac{1}{n}$. In the case of at least one missing block the probability of the same sub-optimal availability is at least $\frac{n}{\kappa+n+1}$.

   Network Coding can solve this problem optimally by choosing $\kappa + n + 1$ independent linear coefficients. This is possible if the base of the underlying Galois field is large enough.

3. The same scenario is used in this case. The optimal policy for FEC($\kappa + 1$) is to transmit $\kappa + 1$ error-correcting code blocks in the first $\kappa + 1$ rounds and n uncoded blocks in the subsequent rounds. Again the availability after the last round is one, since because of $\kappa + 1$ error correction blocks there are neither duplicated nor missing blocks.

□

With FEC(κ) we have a coding scheme that easily outperforms BitTorrent. However this comes at the cost of increasing read/write operations, leading to a cost of up to $O(n^2)$ equaling Network Coding. Since FEC is static, it can be outperformed by Network Coding. In [7] empirical evidence was presented for both performance relations. Good reasons for FEC are the linear efficiency for small κ and the provision of at least some redundancy.

3. PAIRCODING

We have presented Paircoding [10] as a sparse form of Network Coding. Here, each code block is a linear combination of at most two original blocks. The advantage is, that the matrix inversion necessary for decoding can be computed in linear time. Further, only an almost linear number of disk read/write accesses is necessary for decoding. Moreover, the decoding can be done during download, i.e. it continues with each new downloaded code block. In contrast to this, Network Coding can start the decoding process only if all n code blocks have been downloaded already. Paircoding is

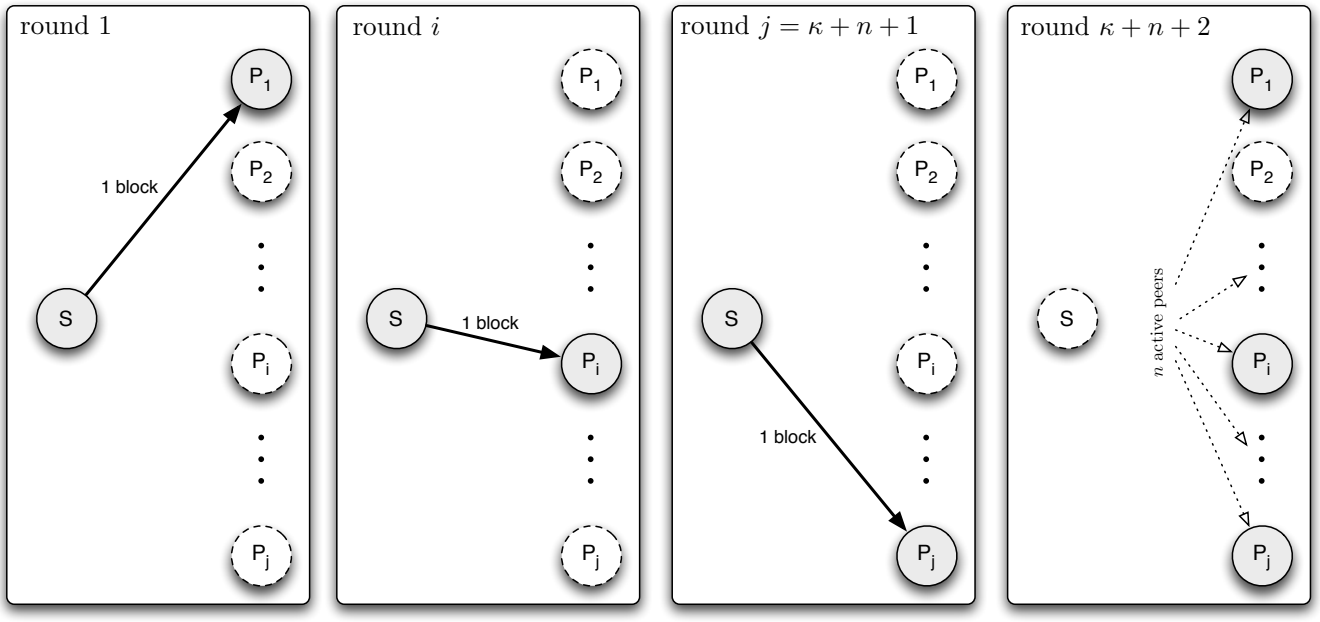

Figure 2: The scenario of the proof of Theorem 1.

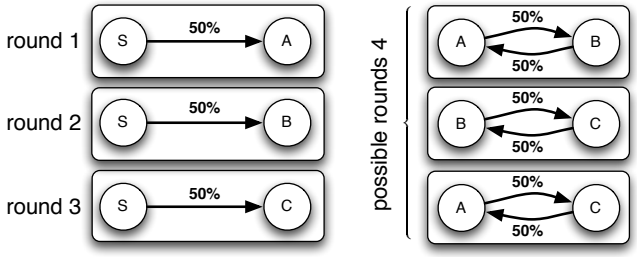

Figure 3: In this scenario Paircoding and Fixed Paircoding outperform BitTorrent and perform as well as Network Coding.

a dynamic coding scheme, i.e. peers can recode new code blocks from other code blocks not decoded yet.

In Paircoding each code block combines at most two blocks x_i, x_j into a code block $b_{\vec{c}}(i,j) = c_1 x_i + c_2 x_j$, where the choice of the positions and (possibly zero) coefficients is described by the policy. Consider a multigraph where the file blocks are the node and code blocks are the undirected edges. Then, a cycle in this graph indicates that all nodes of the connected component can be decoded. Furthermore, if a file block is available, then all connected nodes in the multigraph can also be decoded. So, the Paircoding policies can easily be described by graph properties and operations.

A straight-forward observation is that it is favorable to distribute disjoint pair code blocks such that all file blocks are covered after delivering half of all blocks. Then, any two peers with progress $\frac{1}{2}$ can decode the file and thus have full availability of the file. Using this technique we have proved in [10] the following theorem for the scenario depicted in Figure 3.

THEOREM 2. *[10]*

1. *For some scenarios, Paircoding performs as well as Network Coding.*
2. *Paircoding outperforms BitTorrent:*

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

Paircoding is able to recode blocks within a connected component of the above mentioned multi-graph. While the decoding of the file can be done with linear disk read/write operations, the recoding induces little extra cost for providing union/find like recoding operations. This has lead to the following theorem.

THEOREM 3. *[10] Paircoding has read and write cost of* $O(n \cdot \alpha(n))$*, where* $\alpha(x) = \min\{i \geq 1 : A(i,1) > \log x\}$ *is the inverse of Ackermann's function.*

4. FIXED PAIRCODING

If we restrict Paircoding to code blocks combining only the blocks x_{2i-1} and x_{2i} for $1 \leq i \leq \frac{n}{2}$ we call this coding scheme Fixed Paircoding. In some scenarios it performs as well as Paircoding. We show that it outperforms BitTorrent and that it is outperformed by Paircoding.

THEOREM 4. *Fixed Paircoding outperforms BitTorrent:*

$$\text{FIXEDPAIRCODING} > \text{BITTORRENT}$$

The proof is analogous to PAIRCODING > BITTORRENT as shown in [10].

THEOREM 5. *Paircoding outperforms Fixed Paircoding:*

$$\text{PAIRCODING} > \text{FIXEDPAIRCODING}$$

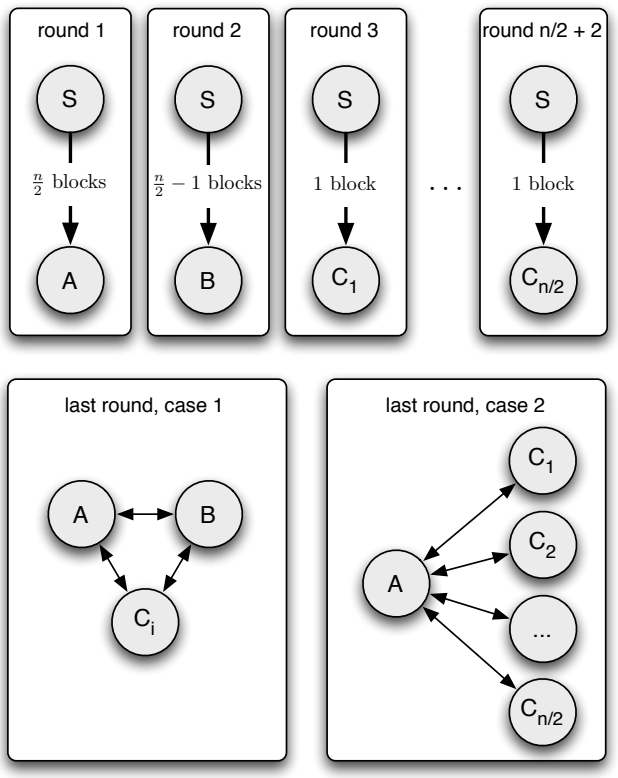

Figure 4: Scenario of the proof of Theorem 5 which proves Paircoding > FixedPaircoding.

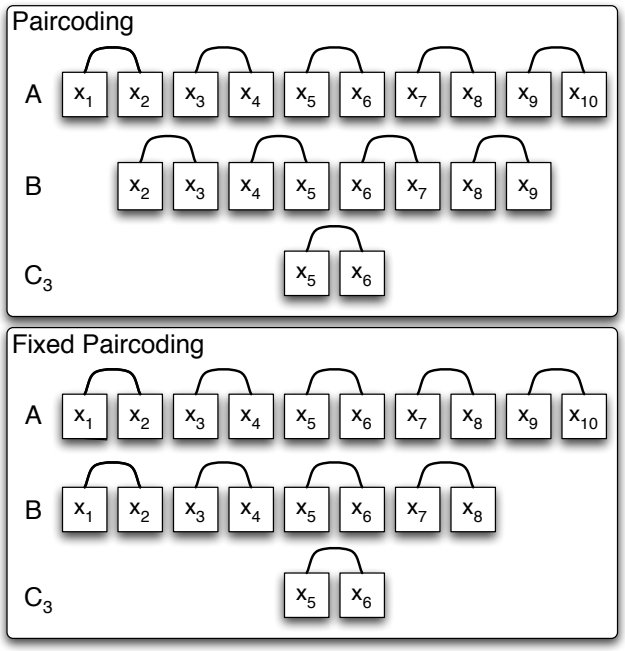

Figure 5: Example file sharing states of the proof of Theorem 5. For both coding systems peer A has received $\frac{n}{2}$ blocks and peer B has $\frac{n}{2}-1$ blocks. Peer C_3 has been randomly chosen from $C_1, \ldots, C_5$. As shown in the proof, Paircoding succeeds in decoding the file while Fixed Paircoding cannot decode the blocks x_9 and x_{10}.

PROOF. Obviously, PAIRCODING $\geq$ FIXEDPAIRCODING, since Fixed Paircoding is a more restricted version of Paircoding.

Regarding outperformance consider the following scenario, Figure 4. In the first round the seed S can upload $\frac{n}{2}$ code blocks to peer A. In the second round the seed can upload $\frac{n}{2}-1$ code blocks only to peer B. In the following $\frac{n}{2}$ rounds the seed can upload only one block to each of the peers $C_1, \ldots, C_{\frac{n}{2}}$. Then with probability $\frac{1}{2}$ the scenario activates peers A and $C_1, \ldots, C_{\frac{n}{2}}$ with unlimited bandwidth. With probability $\frac{1}{n}$ it activates A, B and C_i for $i \in \{1, \ldots, \frac{n}{2}\}$.

First we prove that Fixed Paircoding cannot achieve the full availability in all cases. The only possibility to achieve an availability of one in the first case where A and $C_1, \ldots, C_{\frac{n}{2}}$ are activated in the last round is to send different pairs to the peers $C_1, \ldots, C_{\frac{n}{2}}$. But then only for one of the other $\frac{n}{2}$ cases the availability can be one. This is the case if the missing pair block of B is provided by the active peer C_i. In all other cases the availability is at most $1-\frac{1}{n}$, see Figure 5. Hence, the probability of a failure of any Fixed Paircoding policy is at least $\frac{1}{2}(1-\frac{2}{n})$.

Paircoding can use the following policy to achieve full availability. A receives combinations of blocks $b(2j-1, 2j)$ for $j \in \{1, \ldots, \frac{n}{2}\}$. B receives blocks $b(2j, 2j+1)$ for $j \in \{1, \ldots, \frac{n}{2}-1\}$. Each peer C_i receives block $b(2i-1, 2i)$ with linearly independent coefficients from the blocks of A. In all cases the multigraph consists of one connected component with a cycle, which implies full availability. □

The main advantage of Fixed Paircoding is its low computational cost.

LEMMA 2. *Fixed Paircoding has read/write cost of $O(n)$.*

PROOF. To decode a block x_i, only two different code blocks of form $b(i, i+1)$ with different linear coefficients are required. The same blocks can simultaneously be used to decode x_{i+1}. Thus the total read/write cost are $O(n)$. □

5. TREECODING

Now we define Treecoding which features uncoded blocks, forward error correction blocks, as well as Fixed Paircoding blocks. Here, code blocks are defined by a complete binary tree. We assume that n is a power of two which can be achieved by choosing the block size appropriately and filling up the last block with zeros. The leaves of this tree form the uncoded blocks of Treecoding multiplied by a coefficient. Starting from the second layer code blocks are generated by adding two children code blocks, i.e. computing the vector-wise Xor of the code blocks.

$$
\begin{aligned}
b_i^{\log n}(\vec{c}) &= c_i x_i && \text{for } i \in \{1, \ldots, n\} \\
b_i^{j-1}(\vec{c}) &= b_{2i-1}^{j}(\vec{c}) + b_{2i}^{j}(\vec{c}) && \text{for } j \in \{1, \ldots, \log n - 1\}, \\
& && \quad i \in \{1, \ldots, 2^{j-1}\}
\end{aligned}
$$

The central property of this coding tree is stated in the following lemma which follows directly from the definition.

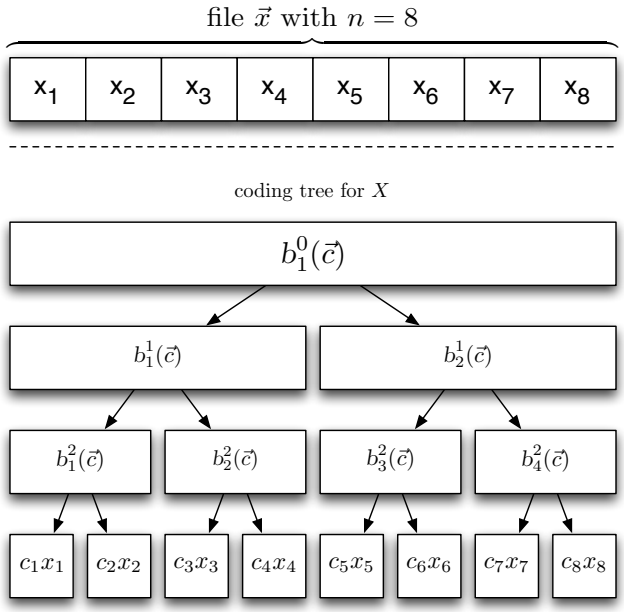

Figure 6: Coding tree of a file X with $n = 8$. The leaves represent the original file blocks $x_1, \ldots, x_8$; they could also be denoted as $b_8^i(\vec{c})$, $i \in \{1, \ldots, 8\}$.

LEMMA 3. *If $\vec{c}$ is known, then any code block (node) in the coding tree can be calculated by a simple Xor operation*

1. *from its two children code blocks, or*
2. *from its parent code block and its sibling.*

A reasonable policy for Treecoding is to start to transmit the root block $b_1^0(\vec{c})$ which is in fact a forward error correction block from a Reed-Solomon code. Then, the policy distributes the children blocks by sending one code block $b_1^1(\vec{c})$ and computing the sibling $b_2^1(\vec{c})$ using the lemma above. This way, the tree is distributed and computed from the root to the leaves by sending exactly n code blocks.

Redundancy in Treecoding is limited, however. To further increase redundancy, we use several coding trees with different coefficients. Especially in highly dynamic scenarios, where downloading peers join and leave the network, multiple coding trees are beneficial. So, a coding tree is defined by the coding vector $\vec{c}$.

Definition 7. If we use Treecoding with at most κ coding trees denoted by the linear coefficients $\vec{c_1}, \ldots, \vec{c_\kappa}$, we call this class TREECODING(κ).

Decoding from multiple coding trees is computationally more complex. This can be seen from the extreme case where n coding trees are allowed and only the root blocks are distributed. Then, Treecoding is as complex as Network Coding. However, for a small number of trees the decoding remains feasible.

THEOREM 6. TREECODING*(κ) has read/write cost of*

$$C_{rw}(\text{TREECODING}(\kappa)) = O(n) \quad \text{if } \kappa = 1 \text{ and}$$
$$C_{rw}(\text{TREECODING}(\kappa)) = O(n^2) \quad \text{for any } \kappa.$$

PROOF. For $\kappa > 1$ the receiver can construct a matrix M of all coefficients of the received tree code blocks. If the rank of this matrix is n the receiver can choose n rows from the matrix and the corresponding code blocks. Likewise in Practical Network Coding the receiver computes the inverse of M and multiplies it with the code blocks which leads to $O(n^2)$ disk read/write operations.

If $\kappa = 1$ then the leaves can be computed by repeatedly using Lemma 3 if the receiver has received enough code blocks with independent coefficients. For one tree this implies that for each node in the codeing tree the sum of all code blocks in the left sub-tree plus the sum of all code blocks in the right sub-tree plus the number of code blocks on the path from the root to the node is at least 2^h where h is the height of the node. From this observation it follows that for all heights h there exists a sub-tree with 2^h code blocks. So, the tree can be decoded with linear disk read/write operations starting from the smallest of these trees, i.e. an uncoded block. □

THEOREM 7. *There exists a performance hierarchy within the Treecoding scheme:*

$$\text{TREECODING}(\kappa + 1) > \text{TREECODING}(\kappa)$$

PROOF. The proof is analogous to the hierarchy of FEC. In the scenario we use $\kappa + n + 1$ rounds to distribute at most one block to $\kappa n + 1$ different peers. Then n random peers of the downloading peers are activated in the last round with unlimited bandwidth.

Treecoding with κ trees can produce at most κ root blocks. We assume that in each round a block is distributed because sending a block is always more advantageous than not sending anything. All other at least $n + 1$ code blocks are either in the left or in the right sub-tree of a coding tree. One side has at least $\frac{n}{2} + 1$ code blocks which follows from the pigeon hole principle. W.l.o.g. this is the left side. Now there is positive probability that all these blocks from the left side and $\frac{n}{2} - 1$ blocks from the right side are chosen. Then, the rank of the matrix is less than n and hence the availability is less than one.

With $\kappa + 1$ trees there is a policy which always works. For this, we distribute all $\kappa + 1$ root blocks and all n leaf blocks. Assume that in the last round $j \geq 0$ root nodes have been chosen. Let ℓ be the missing leaf nodes from the left sub-tree and r the missing nodes from the right sub-tree under the root node. Now, we have $r + \ell + j = n$. So, we use the root blocks as forward error correction blocks to substitute left or right leaf nodes leading to an availability of one. □

THEOREM 8. *Treecoding performs as well as FEC:*

$$\text{TREECODING}(\kappa) \geq \text{FEC}(\kappa)$$

PROOF. To show TREECODING(κ) $\geq$ FEC(κ) we define for every policy in FEC an analogous policy in Treecoding. If FEC decides to send an uncoded block, Treecoding sends the corresponding leaf of a coding tree. If FEC sends the i-th error correction block, then Treecoding sends the root block of the i-th coding tree. Obviously, both strategies are identical. □

We now examine the relationship between Fixed Paircoding and Treecoding with an unlimited number of trees.

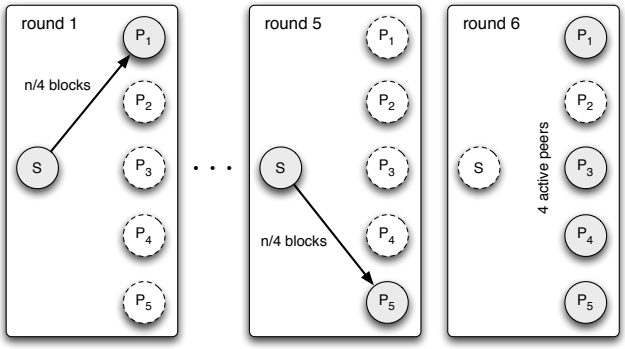

Figure 7: The scenario used in the proof of Theorem 9.

THEOREM 9. *Treecoding outperforms Fixed Paircoding:*

$$\bigcup_{\kappa} \text{TREECODING}(\kappa) > \text{FIXEDPAIRCODING}$$

$$\text{TREECODING}(5) \not\leq \text{FIXEDPAIRCODING}$$

PROOF. For showing TREECODING $\geq$ FIXEDPAIRCODING we can emulate every policy of Fixed Paircoding by choosing leaf nodes or nodes of height one in the coding trees.

Now we show the second claim. We consider a seed and five peers $P_1, \ldots, P_5$. In round i the seed can upload at most $\frac{n}{4}$ blocks to peer P_i. In the last round four random peers from $P_1, \ldots, P_5$ are active without transmission bounds.

Consider a policy for Fixed Paircoding which solves the case that peers $P_1, \ldots, P_4$ are chosen. Then every block pair occurs exactly twice. If this policy chooses at least one block to be transmitted to P_5 this block combines a block pair for the third time. Now consider a case where P_5 is chosen and the two other peers are chosen which also possess that block pair. Then it is overrepresented and there must be a missing code block among the other $\frac{n}{2} - 1$ block pairs. Hence, Fixed Paircoding does not reach full availability in this case.

For Treecoding we choose five different coding trees. We send tree nodes of height two to all peers where each peer receives all $\frac{n}{4}$ tree nodes of the same coding tree. So, any combination of four peers receives the four code blocks for a sub-tree with four nodes. This corresponds to four independent coefficient vectors over these leaf nodes which can be used to decode each original file block.

The second claim now implies the outperformance stated in the first claim. □

COROLLARY 1. *For some scenarios, Treecoding performs as well as Network Coding and better than BitTorrent.*

At the moment the full relationship between Paircoding and Treecoding is unkown. We conjecture that these coding systems do not outperform each other, i.e. for some scenarios Paircoding outperforms Treecoding and vice versa.

CONJECTURE 1. *Paircoding and Treecdoing are incomparable:*

$$\text{PAIRCODING} \neq \text{TREECODING}$$

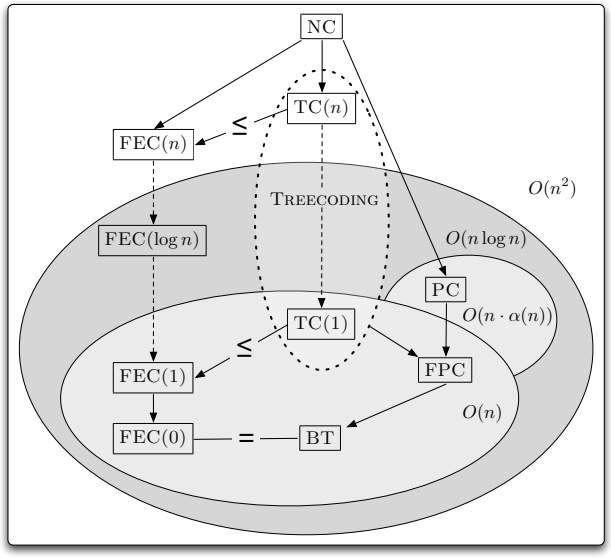

Figure 8: The different coding systems pictured as classes. An arrow indicates the relation "outperforms (<)" of two classes. Dashed arrows indicate a series of relations. The pictured classes are NC: Network Coding, FEC(κ): Uncoded blocks with κ forward error correction blocks, TC(κ): Treecoding with κ different trees, PC: Paircoding, FPC: Fixed Paircoding, and BT: BitTorrent. Also indicated is the order of the decoding read/write cost.

6. SUMMARY AND OUTLOOK

In this paper we compare different new sparse coding classes to Network Coding and BitTorrent. We show that the simple forward error correction scheme FEC(κ) establishes a performance hierarchy between BitTorrent and Network Coding when the number of code blocks increases. We present a restricted version of Paircoding, called Fixed Paircoding, classify its performance, and show that it lies between Paircoding and BitTorrent. Furthermore, we introduce Treecoding and prove its superiority with respect to FEC. Of course, Network Coding remains the optimal performance scheme regarding availability.

However, Network Coding suffers from high disk read/write complexity. While Treecoding and FEC can reduce these costs significantly when error correction is limited, Fixed Paircoding and Paircoding have always linear or almost linear read/write cost. Figure 8 gives an overview of the results in this paper.

One of the main contributions of this paper is the round model which allows to formalize and analytically compare different block based file sharing systems. Up to now the only available method has been empirical experiments using simulations or observations in existing peer-to-peer networks. The presented results indicate that this methodology is manageable and leads to new insights for file sharing systems.

For future research the relationship between Paircoding and Treecoding needs to be investigated as well as other relationships between systems presented here. At the mo-

ment we are preparing papers with simulation results that underline the practical implications of our findings. The main open question is whether an efficient $O(n \cdot \log n)$ or even $O(n)$ disk time file sharing system exists that performs as well as Network Coding. Furthermore, we conjecture for Treecoding that its read/write cost (see Theorem 6) can be improved to $O(\kappa \cdot \log n \cdot n)$ for $\kappa < n/\log n$.

7. REFERENCES

[1] R. Ahlswede, Ning Cai, S. Y. R. Li, and R. W. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204–1216, 2000.

[2] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth content distribution in cooperative environments. In *International Workshop on Peer-to-Peer Systems (IPTPS), LNCS*, volume 2, 2003.

[3] 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)*, 20(8):1489–1499, 2002.

[4] P. Chou, Y. Wu, and K. Jain. Practical network coding. In *Proceedings of the 41st Allerton Conference on Communication, Control, and Computing*, September 2003.

[5] Bram Cohen. Incentives build robustness in BitTorrent. Technical report, bittorrent.org, 2003.

[6] 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. RFC 2362, Internet Engineering Task Force, June 1998.

[7] Christos Gkantsidis and Pablo Rodriguez. Network coding for large scale content distribution. In *INFOCOM*, pages 2235–2245. IEEE, 2005.

[8] Kirsten Hildrum, John D. Kubiatowicz, Satish Rao, and Ben Y. Zhao. Distributed object location in a dynamic network. In *SPAA '02: Proceedings of the fourteenth annual ACM symposium on parallel algorithms and architectures*, pages 41–52, New York, NY, USA, 2002. ACM Press.

[9] Dave Levin, Katrina Lacurts, Neil Spring, and Bobby Bhattacharjee. Bittorrent is an auction: analyzing and improving bittorrent's incentives. *SIGCOMM Comput. Commun. Rev.*, 38(4):243–254, 2008.

[10] Christian Ortolf, Christian Schindelhauer, and Arne Vater. Paircoding: Improving File Sharing Using Sparse Network Codes. *To appear at the fourth International Conference on Internet and Web Applications and Services, ICIW*, 2009.

[11] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in BitTorrent? In *NSDI'07*, Cambridge, MA, April 2007.

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

[13] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Application-level multicast using content-addressable networks. In Jon Crowcroft and Markus Hofmann, editors, *Networked Group Communication, Third International COST264 Workshop, NGC 2001, London, UK, November 7-9, 2001, Proceedings*, volume 2233 of *Lecture Notes in Computer Science*, pages 14–29. Springer, 2001.

[14] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, June 1960.

[15] Antony Rowstron and Peter 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),*, 2218:329–350, 2001.

[16] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In Roch Guerin, editor, *Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01)*, volume 31, 4 of *Computer Communication Review*, pages 149–160, New York, August 27–31 2001. ACM Press.

[17] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. RFC 1075, Internet Engineering Task Force, November 1988.

[18] Mea Wang and Baochun Li. How practical is network coding? *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on*, pages 274–278, June 2006.

[19] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of NOSSDAV*, June 2001.