Rescuing Tit-for-Tat with Source Coding

Thomas Locher, Stefan Schmid, Roger Wattenhofer
{lochert, schmiste, wattenhofer}@tik.ee.ethz.ch
Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 8092 Zurich, Switzerland

Abstract

Tit-for-tat is widely believed to be the most effective strategy to enforce collaboration among selfish users. However, it has been shown that its usefulness for decentralized and dynamic environments such as peer-to-peer networks is marginal, as peers can rapidly end up in a deadlock situation. Many proposed solutions to this problem are either less resilient to freeloading behavior or induce a computational overhead that cannot be sustained by regular peers. In contrast, we retain tit-for-tat, but enhance the system with a novel form of source coding and an effective scheme to prevent peers from freeloading from seeding peers. We show that our system performs well without the risk of peer starvation and without sacrificing fairness. The proposed solution has a reasonably low overhead, and may hence be suitable for fully distributed content distribution applications in real networks.

1 Introduction

Tit-for-tat has become a popular and well-studied strategy since the famous experiment by the sociologist Robert Axelrod [3] who proved its effectiveness in situations where rational players aiming at maximizing their own utility have repeated interactions. In this strategy, a player initially cooperates, and then responds in kind to an opponent's previous action. Tit-for-tat is also applied as a strategy to ensure fairness in distributed networks such as peer-to-peer (p2p) systems which do not possess a centralized control facility, and where the participants can often be regarded as selfish players. For example, tit-for-tat serves as a design principle in the p2p file sharing system *BitTorrent* [6]. In this context, peers follow the tit-for-tat policy if they keep providing parts of a particular file as long as they receive something of interest in return. Generally, fairness in p2p computing is of prime importance, as any system's performance crucially depends on collaboration.

While tit-for-tat is considered to be the most successful strategy to enforce collaboration, there are several problems in practice. In a pure tit-for-tat environment, a peer is only provided with a certain contribution repeatedly if it continues to give something in return. Unfortunately, this is often impossible: In a file sharing environment, a newly joining peer inherently has to download some parts of the file before it is able to upload and hence collaborate with the other peers. Another problem with tit-for-tat in file sharing applications is that a peer's neighbors sometimes have nothing to offer which the peer does not already have, inhibiting any exchanges. Hence, pure tit-for-tat is not a suitable strategy to ensure fairness in dynamic, distributed systems. While systems like BitTorrent have proposed mechanisms to mitigate these problems, the deviation from tit-for-tat implies a loss of fairness and enables malicious peers to freeload [14, 15, 16, 19].

This paper presents and analyzes efficient mechanisms that enforce fairness. In particular, a novel form of *source coding* is introduced which can accelerate tit-for-tat exchanges significantly. We advocate a source coding approach as the resulting data block diversity is large enough for fast content distribution, while the computational burden on the peers is greatly reduced compared to network coding approaches. The bootstrap and seeder problems are addressed as well. Our solution is inspired by the so-called *Allowed Fast* message which is part of BitTorrent's *Fast Extension mechanism*[1]: Newly arrived peers can retrieve a limited amount of encoded packets for free from the seeders. This "seed capital" allows peers to quickly start exchanging data with other peers.

Concretely, in our approach, a distributor or *seeder* divides a file into several blocks, and then—instead of sending the blocks directly and individually—computes a linear combination of a small number k of these blocks. We show that with this technique, the number of exchanges a peer is interested in with its neighbors is considerably larger, and that networks can survive much longer compared to pure tit-for-tat environments without coding. Moreover, due to the small number k, the matrices used to describe the encoded data are sparse.

[1] See http://www.bittorrent.org/fast_extensions.html.

This paper is organized as follows. We will present the main foundations of our approach in Section 2. Section 3 explores the parameter space and discusses possible trade-offs. We further evaluate the performance of our system and the achieved fairness. Related work on existing fairness mechanisms and coding schemes is reviewed in Section 4. Finally, Section 5 concludes the paper.

2 System Overview

We assume that several peers are interested in the same file f offered by one or more peers. The peers still in the process of downloading the file are commonly referred to as *leechers* while the peers possessing the entire file are called *seeders*. Apparently, at least one seeder must be present in the beginning and, while having several seeders is beneficial, our simulations show that typically one seeder suffices to successfully disseminate the content. The file f is divided into m data blocks $b_1, \ldots, b_m$ which are used as the atomic units of trade. These data blocks are not shared directly: The seeders encode them first and offer these encoded blocks to the leechers. How these data blocks are encoded and subsequently decoded by the leechers once they have obtained enough encoded data blocks is described next.

2.1 Source Coding Mechanism

Each data block is considered to be a sequence of elements from a certain alphabet. Typically, in network coding, these elements are elements from a *Galois field* $GF(2^q)$ where q denotes the length in bits of each element in the sequence. In contrast, in our encoding all computations are carried out in a finite field modulo a *Mersenne prime number*. A Mersenne prime number $\mathcal{P}$ is a prime number for which it holds that there is a number $x \in \mathbb{N}$ such that $\mathcal{P} = 2^x - 1$. In our implementation we use the Mersenne prime number $2^{31} - 1$ and a data block size of 128KB, which means that each block consists of a sequence of 33,825 elements from the finite field, as each element requires 31 bits, plus one "parity bit." This last bit in each block is special and will be treated later. Let $b_i[j]$ denote the j^{th} element of the i^{th} data block. When performing basic algebraic operations (such as addition and subtraction) on data blocks, the operations are performed on all the elements in the blocks separately, i.e., when an operation $\diamond \in \{+, -, \times, \div\}$ is applied to data blocks b_x and b_y, we get the data block b^* for which it holds that $b^*[i] = b_x[i] \diamond b_y[i]$ for all i. The reason we use Mersenne prime numbers is that these numbers are almost a power of two, which allows for efficient implementations of the basic algebraic operations. In fact, the multiplication and division operations are similar to the corresponding operations in a Galois field $GF(2^q)$; for instance, when computing the inverse of an element, the *extended Euclidean algorithm* can be applied. In contrast to systems based on $GF(2^q)$ where addition and subtraction are both XOR operations, we perform regular addition and subtraction operations modulo $\mathcal{P}$ in our system. Because $\mathcal{P}$ is a Mersenne prime number, these operations can be performed fast: Whenever the result of an addition exceeds $\mathcal{P}$, which means there is a "carry bit," this bit can simply be dropped and the result incremented by one.[2]

Seeders never distribute data blocks directly, they compute additions (or *linear combinations*) of exactly k randomly chosen data blocks for some small number k (the optimal parameter k will be determined later) and advertise these blocks. A leecher downloading such a block additionally needs to download a vector containing the information which blocks have been added up to form this new block. Even for small k, the *block diversity* increases substantially compared to systems which do not use coding, ensuring that peers do not starve and do not run into a deadlock situation when exchanging blocks according to the tit-for-tat policy. The following simple example motivates this point. Consider two peers, p_1 and p_2, both of which already have downloaded $m/2$ blocks from distinct sources, and assume the downloaded subsets of all blocks are more or less random. If no coding is used, i.e., $k = 1$, p_1 is expected to be interested in about half of the blocks p_2 has already downloaded and vice versa. They will thus play tit-for-tat until they both acquire approximately $3m/4$ blocks after which they will need new peers to share their data blocks with. In case source coding is used, the space of possible blocks is considerably larger. Peer p_1 is interested in about $m/2\big(1 - m/\left(2\binom{m}{k}\right)\big)$ blocks possessed by p_2, and hence the two peers are able to almost finish their download with tit-for-tat without the help of any additional peers, even for small k.

In order to reconstruct the original data blocks, a peer has to download at least m—ideally *exactly* m—encoded blocks which are all created from different linear combinations of the original blocks. The original blocks are then retrieved by solving a linear system of equations. Let $\mathcal{C}$ denote the coefficient matrix containing the information which blocks have been used to form each downloaded block. Note that the coefficient matrix $\mathcal{C}$ is *sparse*: In each row, there are precisely $k \ll m$ ones, and the rest are zeros. More importantly, observe that the original data can be reconstructed from this linear system if and only if the rank of $\mathcal{C}$ is m. Simulations reveal that the rank of such a matrix is basically *always* less

[2] Of course, there is a small overhead whenever the result has to be incremented by one. According to our simulations, this does not happen exceedingly often and it consequently does not have a noticeable impact on the running time.

than m if XOR is used as the addition operation. This problem can be circumvented by multiplying each block with a random weight before adding them up which results in a matrix that has a full rank with high probability; such weights are commonly used in network coding. However, in our system, by performing regular addition operations modulo $\mathcal{P}$ rather than modulo 2, the rank is practically always exactly m. In the decoding phase, the matrix $\mathcal{C}$ is inverted and this inverse matrix is then used to solve the linear system of equations. Computing the inverse first is much more efficient because it does not involve any data blocks, and the subsequent computation to retrieve the original data blocks requires $O(m^2)$ operations.

Our source coding technique has several interesting properties. First, there is no need for weights which simplifies the system and yields efficient block generation operations. Second, because each block is either used in a linear combination or not, a simple bitmap as a characterization of each encoded block suffices, while a vector containing all weights have to be sent in case weights are used. Third, due to the simple structure of the coefficient matrix, the matrix can be inverted quickly.

There are some disadvantages of our approach which have to be addressed. As computations are performed modulo $\mathcal{P} = 2^x - 1$, the bit pattern consisting of x ones cannot be distinguished from x zeros. Moreover, the last bit in each block cannot be reconstructed using the matrix, because the matrix does not have a full rank when calculating modulo 2 as mentioned before. Our solution is to create a *helper block* at the seeder containing the positions where x consecutive bits set to 1 appear in the file and also the last bit from each block. Typically, in compressed files, such sequences of length x (31 in our case) are scarce. Storing the last bit of every block is also cheap, e.g., when using a block size of 128KB, only 1KB is needed to store the last bits of each block of a 1GB file.

While the trades between leechers are fair using tit-for-tat because every peer is required to exchange blocks in order to acquire other blocks, peers still have to be prevented from exploiting the benevolence of seeders which merely remain in the system to assist leechers in their effort to acquire the missing blocks. Thus, there is a need for a suitable seeder strategy that effectively helps peers, but successfully undermines freeloading behavior.

2.2 Seeder Strategy

Both source and network coding are powerful techniques to improve the efficiency of a content distribution system for example in terms of bandwidth usage and average download completion time. We use source coding primarily to increase the data block diversity and thereby establish fair trading of resources without the risk of deadlocks. Likewise, our seeder strategy is based on a strategy whose main purpose is to boost the download progress by providing newcomers with initial data they can share. The *Allowed Fast* message of BitTorrent's Fast Extension mechanism defines for any peer a small but specific pseudo-random set of data blocks that this peer can download immediately for free, i.e., without uploading anything in return. This set is created according to the peer's IP address and hence all seeders produce the same free set for a given requesting peer. Consequently a peer is not able to get additional blocks for free by inquiring another peer or by reconnecting to the same seeder at a later time.

Our system adapts this idea to provide an initial pseudo-random set of encoded blocks to all peers. In contrast to the original Fast Extension mechanism, only the seeders compute such sets of blocks since the leechers by definition do not possess all blocks, and the seeders are the only peers giving out blocks from such sets. If a leecher connects to a seeder, the seeder will announce these downloadable linear combinations. Another alteration of the original mechanism is that the seeders continuously modify the size of the allowed subset of linear combinations according to the number of peers in the network. If there are approximately n peers in the network, the number of free sets can be set to $\min\{m, \max\{\alpha, \beta \cdot m/n\}\}$ where α and β are small positive constants greater than 1. The seeders do not need to compute the number of peers in the network, its own number of connections to leechers can be used as an estimate.

The combinations a newly arrived peer receives from the seeder forms the starting set. The chances that any neighboring peer already possesses one of these linear combinations in its starting set are small, implying that many data blocks can be exchanged. Apart from these pseudo-random but individually well-defined sets of blocks, seeders never contribute any other parts of the file.

The benefits of this mechanism are threefold. First, peers with different IP addresses obtain entirely different encoded blocks. This ensures that the available number of distinct linear combinations in the network is high. Second, the leechers are forced to collaborate because the seeders merely provide a small and specific set of blocks to each peer and refuse to provide any other blocks. It is thus in any peer's best interest to upload to other leechers as much as possible, resulting in an efficient bandwidth usage in the network. Moreover, it is unlikely that seeders are inundated with requests from leechers as the seeders only provide little data. A seeder can thus stay in the network without having to sacrifice a large fraction of its upload capacity. Third, as the size of the free set depends on the number of peers in the network, the system's resiliency to freeloaders increases as the network grows. Furthermore, if there are only a few interested

peers, each of them gets a reasonable share of the file for free so that they can all finish their downloads by exchanging their blocks amongst each other.

As a final remark, we note that it is clearly more challenging to ensure *data integrity* in such a system because it is unfeasible to store and provide hashes for all possible $\binom{n}{k}$ encoded blocks. However, expedient solutions to this problem have already been presented in the literature. For example, when using *homomorphic hash functions* [9], merely hash values for each original data block must be stored, and the hash codes of all linear combinations can be computed using the hashes of the original data blocks. As these hash functions are computationally expensive, another scheme called *secure random checksums* [8] based on random masks and mask-based hashes has been devised. These secure random checksums can be computed very efficiently and any intended or unintended corruption of data blocks can be detected. Thus, a large data block diversity can be achieved without compromising the protection against the insertion and distribution of polluted data blocks.

3 Evaluation

In the first part of this section, appropriate values for the number k of original blocks used to create encoded blocks are derived. As the usability of the system depends on the efficiency of decoding the obtained blocks, we further analyze the running time of the decoding phase and point out how to keep the decoding time within reasonable bounds even for large files of several gigabytes. We have further performed simulations of the entire proposed protocol in different scenarios.

3.1 Parameter Space

It is essential to set the parameter k to an adequate value. As the block diversity is already large enough to ensure active and continuous trading when k is about 2 or 3, and both the encoding and the decoding procedures can be performed more efficiently when k is small, it is desirable to minimize k. However, k has to be large enough to ensure that the coefficient matrix $\mathcal{C}$ has full rank after m blocks have been received.

The analysis of the rank of $\mathcal{C}$ is related to the classic *coupon collector problem*. However, in our system, a peer never downloads the same linear combination twice. Moreover, we use $k > 1$, and compute our matrices modulo a large prime. These differences render a precise analysis of the rank quite challenging, and to the best of our knowledge, this problem is not studied in mathematical literature (for more information, cf Section 4).

In order for the matrix to have full rank, it is necessary (but not sufficient) that each column have at least one positive entry. For a given column, the probability that it contains only zeros is $(1-k/m)^m$, so roughly e^{-k} for large m. This implies that k must grow at least logarithmically in m, such that there are $m \cdot e^{-(\log m + \omega(1))} = o(1)$ columns filled with zeros in expectation (for $k > 1$). Cooper [7] considers a model where the non-trivial matrix entries are chosen uniformly at random from a given group (rather than having 1s all the time). Each entry is chosen independently at random. In our case, k has to be slightly larger than $\log m$ for the matrix to have full rank. Unfortunately, it is not clear how these results can be adapted for our model.

We have performed several simulations to analyze the dependence of the rank on k. When setting k to a number slightly larger than $\log m$ (we used $\log m + 2$ in our experiments), we have never encountered an example matrix with a rank lower than m for any reasonably large m. This indicates that the necessary condition of having a positive entry in every column is also sufficient in practice.

3.2 Decoding Phase

Increasing the block diversity by coding implies an additional computational burden as peers are forced to decode the blocks to obtain the desired file. Once the inverse of the coefficient matrix $\mathcal{C}$ has been computed in $O(m^3)$ time, the inverse is multiplied with each encoded block, requiring $O(m^2 s)$ time where s denotes the number of symbols or elements in each data block (33,825 in our implementation). Since s is typically larger than m, the reconstruction time is dominated by the time required to multiply the inverse with all blocks as shown in Figure 1.

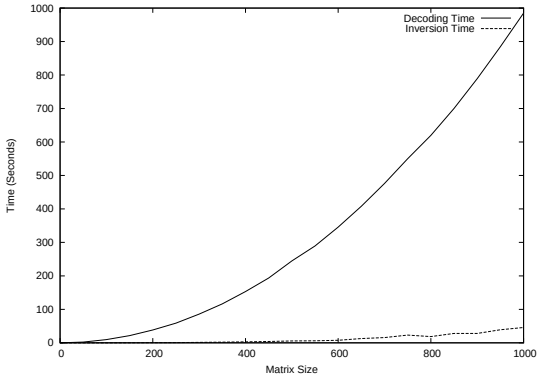

Figure 1. The decoding time exceeds the times to invert the matrices for large $m \times m$ matrices. The data block size is 128KB in all computations.

These computations have been performed on a Pentium Core 2 Quad desktop PC with 4GB of RAM. We decided to test our implementation on such a computer as its computational power reflects the performance we can expect from regular PCs and thus from typical peers in the near future.

In order to quantify how the system benefits from the sparsity of $\mathcal{C}$, we also measured the time it takes to invert full matrices containing random entries from the same field. For a matrix of size $m = 512$, the average time to invert a full matrix is 6.6s, whereas a sparse matrix of the same size with $k = 11$ ones in each row can be inverted in 3.9s. The difference becomes increasingly more pronounced as the matrices grow: Inverting a full $1,024 \times 1,024$ matrix takes roughly $15s$ longer (55.1s vs. 40.4s), implying that there is a certain, albeit small, computational gain in keeping the matrix sparse.

It is apparent from Figure 1 that m cannot be much larger than 1,000, because the decoding phase would become unacceptably long. There are several ways to cope with larger files. A simple solution would be to increase the block size to keep the size of the matrix bounded. However, choosing a larger block size has the disadvantage that it is more likely that the (atomic!) transfers fail, and of potentially large waiting periods due to slow uploading peers. Moreover, a selfish peer can get one atomic transfer unit for free from every peer, and if this unit size is too large, freeloading becomes possible again. A better approach is to encode several blocks together to form one entry in the matrix. Such a group of blocks is also referred to as a *generation*. If g blocks together form one entry in the matrix, the running time of the decoding phase is reduced to $O((m/g)^3 + (m/g)^2 sg)$. For example, if $g = 16$ and the block size is again 128KB, encoding a 1GB file results in a matrix of size $m = 512$, and the running time is roughly 16 times the running time of the decoding of a file of size 64MB without the use of generations.

The drawback of this approach is that peers are forced to interact with specific peers until all blocks from a certain group are obtained. Although all blocks can still be verified individually, all downloaded blocks are useless if merely one block of this group is missing. Thus, while the use of generations substantially reduces the reconstruction time, g should be kept small as large generations potentially constrain the communication in the network.

3.3 Simulation

We now analyze the behavior of the entire system in dynamic environments. Simulations for two different scenarios have been performed:

1) In the first scenario, there is one seeder which is always active. Up to 2,000 peers[3] arrive as follows: In the beginning, there is a rush period (*flash crowd*) where many peers join almost instantaneously. After this phase, the peer arrival rate decreases steadily. Concretely, peer i arrives according a *Poisson process* with parameter $\lambda_i = 10/(2000 - i)$ hours. Peers leave the network again as soon as enough blocks are downloaded.

2) In the second scenario, the seeder leaves the network immediately after it has pushed $4m$ linear combinations into the network. The arrivals and departures of the other peers are as described above.

In our simulations, leechers maintain up to 10 connections simultaneously and drop old connections in favor of connections to new peers. The single seeder accepts connections to up to 50 leechers at any given time, but only provides a few data blocks before dropping the connections again. Overall, the network resembles a typical BitTorrent swarm with the main difference that blocks are exchanged according to our own policy.

For our tests, we assumed a file consisting of $m = 1,024$ blocks, each being of size 128KB. The source coding was performed by combining $k = 12$ blocks. We assigned each peer an upload bandwidth which we chose uniformly at random, and slightly larger downstream bandwidths. In reality, peers are often behind routers using *network address translation* (NAT) and cannot receive incoming connections. Hence, we have also experimented with different percentages of peers with restricted connectivity.

In the first scenario, where the seeder stays online, all peers always find new blocks to download and are consequently able to finish their downloads quickly. Notably, the second scenario works reliably as well, and the network is active much longer than the seeder as the tit-for-tat exchanges alone keep the file distribution going. Figure 2 compares the diversity of all available linear combinations in the network for these two scenarios. As expected, the constant seeder scenario features a substantially larger diversity. But also the diversity of the second scenario deteriorates slowly; peers become stuck only towards the end when no new peers join and a considerable share of all peers have already completed their downloads and left the network.

Our simulations confirm the observation made in [12] that using the tit-for-tat strategy entails a high correlation between upload and download rates, as depicted in Figures 3 and 4. The relation between upstream and download duration is almost ideal, except for the case of a large number of firewalled peers, because these peers fail

[3]This number of peers is what we can typically expect in a large BitTorrent swarm.

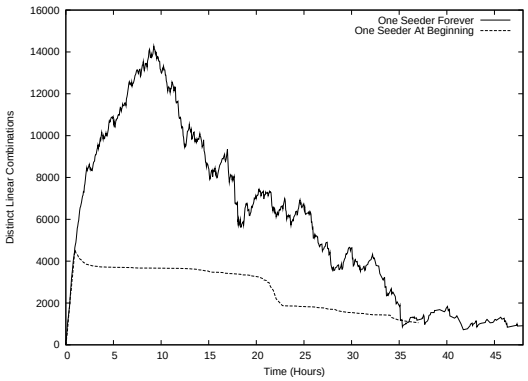

Figure 2. The number of distinct linear combinations available in the whole network. A higher diversity facilitates a more effective tit-for-tat trading of blocks.

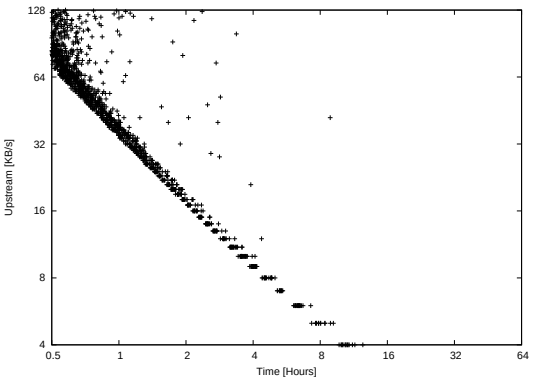

Figure 3. This figure plots the completion times of the peers' downloads. Peers having larger upload bandwidths finish faster than peers contributing less. The correlation is almost linear, with the cloud in the upper left corner being due to the 10% firewalled peers.

to open a sufficient number of connections and therefore need more time.

In our tests, the ratio between the amount of data sent by the average peer and the size of the shared file was usually between 1.1 and 1.15—a reasonably low overhead.[4]

We also compared our system to a pure tit-for-tat approach which does not use coding. We used the *rarest first* block selection strategy, where a peer fetches the block from the set of blocks it is interested in which is found least often in its neighborhood. We have analyzed a network of 300 peers which again join according to a Poisson distribution $\mathcal{P}o(\lambda_i)$ where $\lambda_i = 10/(300 - i)$ hours. We added one seeder at the beginning which leaves the network after having uploaded $4m$ blocks. Note that even if the total number of peers is merely 300, the seeder's free contribution of $4m$ blocks constitutes only a fraction of roughly 1% of the entire data load, which means that almost all data blocks are exchanged using the tit-for-tat policy. The peers leave the network as soon as they have downloaded the entire file.

Figures 5 and 6 depict the difference between the two schemes: In the coding case 254 peers finish their download while in the non-coding case, merely 16 peers can download the entire file. In Figure 5, the seeder left the network after 0:58h, and in Figure 6 after 1:01h. In the latter case, only 17 minutes after the seeder has left the network, the performance degraded severely. In contrast, our system continued to work for 22 more hours.

[4]This calculation does not include TCP/IP overhead but it includes all transmissions by the user, such as handshake messages, requests, etc.

Several additional simulation runs with different parameters confirmed this outcome: Source coding effectively helps in keeping a network diverse. Of course, the parameter of the Poisson arrival process influences the outcome crucially: If the peers arrive at short intervals, the performance is excellent; however, when the interval times increase, the system suffers as the leaving peers take useful knowledge with them which cannot be replaced.

4 Related Work

The existence of free riders in p2p systems has been pointed out in several papers, e.g., see [1, 11] for studies of freeloading in the Gnutella system. Various mechanisms to ensure fair trading in peer-to-peer networks have been proposed, e.g., KARMA [20] and EigenRep [13]. However, many of these mechanisms are complex and introduce an unsustainable overhead. The BitTorrent protocol incorporates simple countermeasures against freeloading. Originally, BitTorrent used tit-for-tat as the underlying strategy for piece exchange. Since pure tit-for-tat turned out to be inoperative in highly dynamic networks—this fact is reconfirmed in this work—the protocol has been modified repeatedly. Due to significant changes such as the introduction of *optimistic unchoking* the current BitTorrent protocol only vaguely resembles the tit-for-tat mechanism. It has been shown recently that BitTorrent does not deter selfish users from free riding [14, 15, 16, 18], and different strategies on

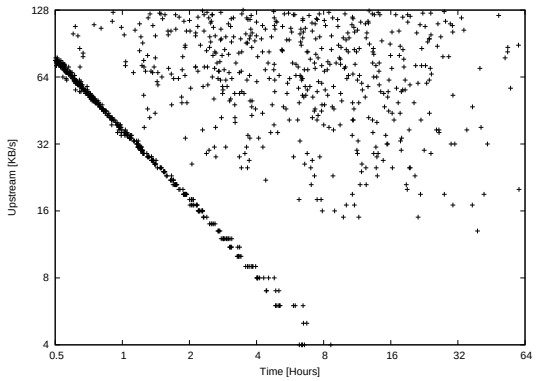

Figure 4. Experiment with 60% firewalled peers. The cloud on the upper right is larger: Since peers behind NATs and firewalls cannot be contacted from outside, less connections are established and the performance suffers. For the non-firewalled peers, the correlation between upload bandwidth and completion time is roughly linear.

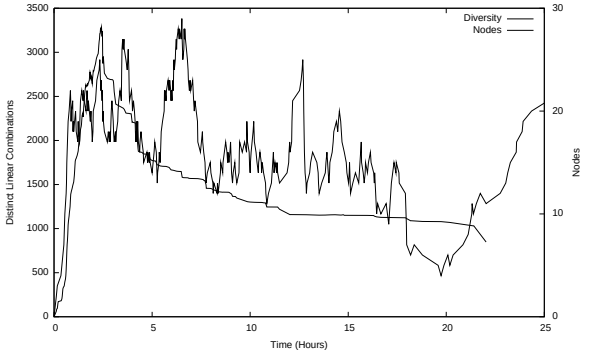

Figure 5. Simulation of content distribution using source coding with $k = 12$. Although the seeder left after 58min, 254 peers are able to download the entire file. Moreover, even after 22 hours, the trade is still active.

how to ameliorate the resistance to free riders have been presented, e.g., [17]. In contrast to many of these works, our approach does not alter the basic tit-for-tat mechanism, but augments the piece exchange mechanism with source encoding and applies a new strategy for seeding peers which have no interest in downloading.

Coding, especially *network coding* [2], has received a lot of attention in the last few years. Chou et al. present a practical network coding system for streaming content in [5], and analytical evidence supporting the effectiveness of coding in peer-to-peer networks is provided in [8]. Gkantsidis and Rodriguez use these results in their proposal of a new scheme for large scale content distribution [10]. In their paper, the authors show that network coding, in which each node creates new linear combinations of blocks to be shared, allows for simple scheduling algorithms and more efficient content distribution. It is further pointed out that peers finish their downloads even if a single seeding peer leaves shortly after uploading merely one copy of the file to the system and all peers selfishly depart after they finish their downloads. This is also true if the tit-for-tat mechanism is applied, hence network coding can be used to guarantee fairness in a peer-to-peer system. Experiences made with a real implementation of their proposed system are presented in [8]. In their system, peers are constrained to maintain only 4-8 connections to other peers. Due to this limitation, network coding outperforms source coding because for any peer the chances that there are permanently blocks of interest in its restricted neighborhood are smaller. The focus of this paper is on *guaranteed fairness* rather than on content distribution *efficiency*. We show that by allowing leechers to open up merely 10 connections, which is a moderate number for content distribution networks, our source coding scheme manages to distribute the blocks efficiently. More importantly, we provide evidence that our coding scheme works reliably when tit-for-tat is used even if seeders only provide a small number of blocks and refuse any further service, demonstrating the practicability of a pure source coding scheme. In contrast to network coding solutions, the computational overhead of the leechers is reduced, as blocks of data are only encoded at the seeders. At the same time, block diversity is still high, enabling efficient tit-for-tat exchanges and hence making the system appealing for practical use.

To the best of our knowledge, there is no other work proposing a similar coding scheme. More generally, it seems that the properties of the matrices produced by our system are hardly explored, and hence, future research on these matrices may help to further speed up our decoding operations. For random matrices as they appear in the study of *random graphs* [4], the situation is different, and there exists a large body of literature. For example, Cooper [7] has studied the rank of square random matrices over $GF(2)$ where each entry is independently and uniformly distributed. We conjecture that some of these results may be adapted for our model.

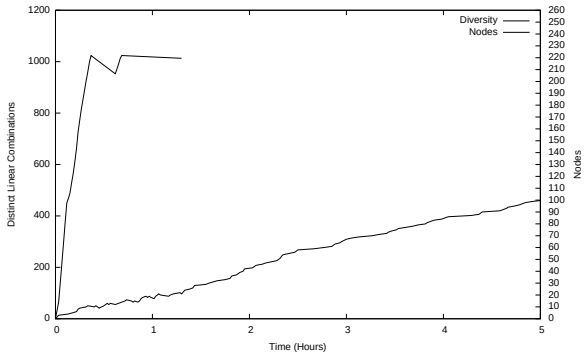

Figure 6. A pure tit-for-tat system without coding. Only 17min after the seeder has left, the piece diversity declines sharply, and most of the peers are not able to complete their downloads.

5 Conclusion

An interesting approach to increase fairness in peer-to-peer systems is the use of coding techniques, and there seems to be a challenging trade-off between fairness and computational complexity. This paper advocates a novel source coding scheme which seeks to minimize the computations performed by the peers and in which only the seeders are involved in the encoding process. We provided evidence that the overhead of our solution is tolerable and that fair tit-for-tat exchanges continue even after all seeders have left, especially if peers have a reasonable number of neighbors. In addition, we presented a strategy which prevents free-riding and enforces collaboration in spite of the presence of seeders which can easily be exploited in protocols such as BitTorrent. Finally, we believe that there are means to further improve the efficiency in future research.

Acknowledgments

We are indebted to Patrick Moor for numerous helpful discussions and for running the simulations. We would also like to thank Reto Spöhel for pointing us to [7], and Thierry Dussuet for his assistance during the preparation of the paper. Our research is supported by the Swiss National Science Foundation project "Decentralized Internet Working."

References

[1] E. Adar and B. Huberman. Free Riding on Gnutella. *First Monday*, 2000.

[2] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network Information Flow. *IEEE Transactions on Information Theory*, 2000.

[3] R. Axelrod. The Evolution of Cooperation. *Science*, 211(4489):1390-6, 1981.

[4] B. Bollobas. *Random Graphs (2nd Edition)*. Cambridge University Press, 2001.

[5] P. Chou, Y. Wu, and K. Jain. Practical Network Coding. In *Proc. 51st Allerton Conference on Communication, Control and Computing*, 2003.

[6] B. Cohen. Incentives Build Robustness in BitTorrent. In *Proc. 1st Workshop on Economics of Peer-to-Peer Systems (P2PEcon)*, 2003.

[7] C. Cooper. On the Rank of Random Matrices. *Random Struct. Algorithms*, 16(2):209–232, 2000.

[8] C. Gkantsidis, J. Miller, and P. Rodriguez. Comprehensive View of a Live Network Coding P2P System. In *Proc. 6th ACM SIGCOMM on Internet Measurement (IMC)*, pages 177–188, 2006.

[9] C. Gkantsidis and P. R. Rodriguez. Cooperative Security for Network Coding File Distribution. In *Microsoft Research Tech Report MSR-TR-2004-137*, 2004.

[10] C. Gkantsidis and P. R. Rodriguez. Network Coding for Large Scale Content Distribution. In *Proc. IEEE INFOCOM*, 2005.

[11] D. Hughes, G. Coulson, and J. Walkerdine. Free Riding on Gnutella Revisited: The Bell Tolls? *IEEE Distributed Systems Online*, 6(6), 2005.

[12] S. Jun and M. Ahamad. Incentives in BitTorrent Induce Free Riding. In *Proc. 3rd ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, 2005.

[13] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The Eigentrust Algorithm for Reputation Management in P2P Networks. In *Proc. 12th International Conference on World Wide Web (WWW)*, pages 640–651, 2003.

[14] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploiting BitTorrent For Fun (But Not Profit). In *Proc. 5th Itl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[15] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free Riding in BitTorrent is Cheap. In *Proc. 5th Workshop on Hot Topics in Networks (HotNets)*, 2006.

[16] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkatarami. Do Incentives Build Robustness in BitTorrent? In *Proc. 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, 2007.

[17] S. Sanghavi and B. Hajek. A New Mechanism for the Free-rider Problem. In *Proc. 2005 ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PEcon)*, pages 122–127, 2005.

[18] J. Shneidman, D. C. Parkes, and L. Massoulié. Faithfulness in Internet Algorithms. In *Proc. ACM SIGCOMM Workshop on Practice and Theory of Incentives in Networked Systems (PINS)*, pages 220–227, 2004.

[19] M. Sirivianos, J. H. Park, R. Chen, and X. Yang. Free-riding in BitTorrent Networks with the Large View Exploit. In *Proc. 6th Itl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[20] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. KARMA: A Secure Economic Framework for P2P Resource Sharing. In *Proc. Workshop on Economics of Peer-to-Peer Systems (P2PEcon)*, 2003.