Boosting Market Liquidity of Peer-to-Peer Systems Through Cyclic Trading

Raphael Eidenbenz[1], Thomas Locher[2], Stefan Schmid[3], Roger Wattenhofer[1]
[1] ETH Zurich; {eidenbenz,wattenhofer}@tik.ee.ethz.ch
[2] ABB Corporate Research; thomas.locher@ch.abb.com
[3] Telekom Innovation Laboratories & TU Berlin; stefan@net.t-labs.tu-berlin.de

***Abstract*—Tit-for-tat trading lies at the heart of many incentive mechanisms for distributed systems where participants are anonymous. However, since the standard tit-for-tat approach is restricted to bilateral exchanges, data is transferred only between peers with direct and mutual interests. Generalizing tit-for-tat to multi-lateral trades where contributions can occur along *cycles of interest* may improve the performance of a system in terms of faster downloads without compromising the incentive-compatibility inherent to tit-for-tat trading. In this paper, we study the potential benefits and limitations of such a generalized trading in swarm-based peer-to-peer systems. Extensive simulations are performed to evaluate different techniques and to identify the crucial parameters influencing the obtainable throughput improvements and the corresponding tradeoffs. Moreover, we discuss extensions for overhead reduction and provide an optimized distributed implementation of our techniques. In summary, we find that allowing inter-swarm trades on short trading cycles can improve the throughput significantly; on the other hand, trading on long cycles does not pay off as the communication and management overhead becomes exceedingly large while the additional performance gains are marginal.**

I. Introduction

In most economic systems, goods are exchanged for money. The main reason is that a well-accepted currency helps to overcome the typical problem of barter systems, namely that two individuals need to find matching amounts of exactly the right goods. Money provides *flexibility* as any service offered can be redeemed with any other person who accepts money. Moreover, money provides *temporal freedom* in that a surplus from imbalanced bartering can be stored and redeemed in future transactions. However, barter continues to exist in different forms, e.g., corporate barter, neighborhood barter markets,[1] or organ donation barter.[2]

The largest barter markets today are probably Internet-based, i.e., the Internet serves as a catalyst and platform for various forms of barter-based trading. In fact, a large fraction of the Internet traffic is caused by peer-to-peer file sharing systems that use bartering to exchange data. BitTorrent, arguably the most popular file sharing protocol, is also based on this trading principle: Peers interested in the same content form a so-called *swarm*, and blocks of this content are exchanged directly among peers in the swarm. Since only direct barter is used on a per swarm basis, this policy is called *intra-swarm trading*. The main drawback of this policy is that a peer is not interested in trading with another peer that possesses a subset of its own blocks; this can severely reduce the market liquidity and thus the overall throughput. Measurements [1] show that about half of the peers in BitTorrent are active in multiple swarms at the same time, as illustrated in Figure 1. Hence, *inter-swarm trading* may result in a higher system throughput. In this paper, we study to what extent we can boost the market liquidity of a peer-to-peer system by allowing multi-lateral bartering *across swarms*, without compromising the basic tit-for-tat incentive mechanism.

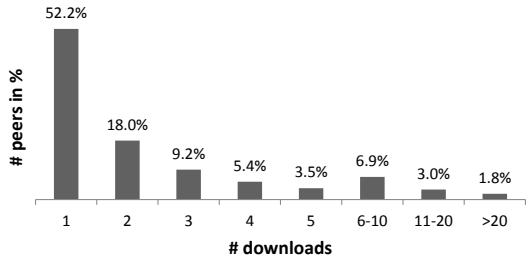

Fig. 1. Distribution of the number of downloads per peer in BitTorrent.

Apart from direct inter-swarm trading, further trading opportunities can arise when peers are allowed to trade along *cycles of interest* (cf. Figure 2). Our measurements in live BitTorrent swarms reveal that even on a subset of the BitTorrent system, the number of such *trading cycles* is large. Moreover, the number of cycles grows fast with increasing cycle length: each additional hop increases the number of cycles by 3-4 orders of magnitude, as shown in Figure 3. In the remainder of this paper, we will refer to a trading cycle consisting of k peers as a *k-cycle*. Figure 4 further illustrates the potential of trading on cycles as it shows that a random peer is part of a large number of inter-swarm cycles with a probability of roughly $1/3$ even if we only consider a subset of 1000 swarms.

A. Our Contributions

This paper addresses the question of how the use of inter-swarm trading cycles affects the achievable throughput in a system. While it is clear that in the best case, the relative throughput increase can be unbounded (in particular when there are no intra-trading opportunities), we focus on the

[1]See for instance http://www.huffingtonpost.com/kirsten-dirksen/barter-markets-can-tradin_b_255545.html for a report on barter markets in Barcelona.

[2]Since monetary trade with human organs is prohibited by law, economists have developed organ donation markets that rely on direct—or even multi-lateral—trades (see, e.g., the 2007 Nobel Memorial Prize in Economics).

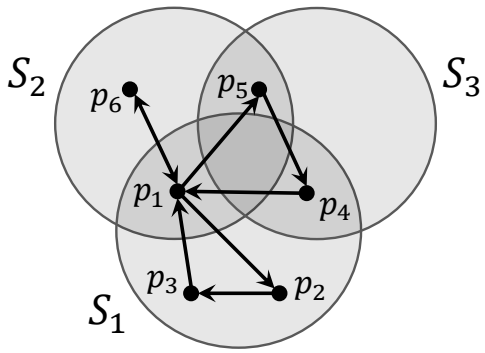

Fig. 2. Possible trading situation for a peer p_1 participating in two overlapping swarms S_1 and S_2: in addition to the direct bilateral trades within a swarm (e.g., with peer p_6 in S_2), p_1 can exchange data along intra-swarm cycles (e.g.,(p_1, p_2, p_3, p_1) in S_1). Moreover, it can trade pieces with one or more peers in different swarms (e.g., along the cycle (p_1, p_5, p_4, p_1)).

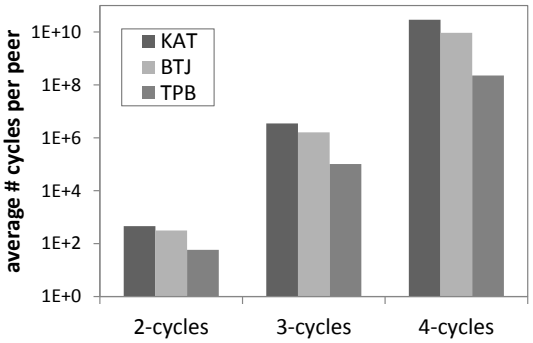

Fig. 3. Average number of inter-swarm k-cycles (cycles of length k) per peer for $k = 2, 3, 4$ for data sets of the top 1000 video torrents of the three torrent discovery sites with the highest Alexa rank, `ThePirateBay.org` (TBP), `BTJunkie.org` (BTJ), and `kat.ph` (KAT). The number of 4-cycles is approximated within an error margin of 1%. Note the logarithmic scale.

throughput gain in practical scenarios. Based on data collected from real swarms we conduct simulations to study the achievable throughput under different trading strategies and in different scenarios.

The results indicate that the throughput of a peer-to-peer system can indeed benefit from trading on cycles. Interestingly, these benefits are obtained already for fairly short cycles, involving up to three nodes. Longer cycles do not lead to a substantially larger performance but incur a large communication and computational overhead. For our evaluation, we also derive a model for the peers' download preferences combining preferential-attachment and co-occurrence principles. Moreover, we identify certain pitfalls, such as the problem of redundant downloads, and we propose techniques that considerably mitigate this problem. Methods to reduce the communication overhead are also discussed. Finally, we outline a distributed implementation of our techniques.

II. Related Work

The peer-to-peer paradigm relies on the contributions of the participants, i.e., the peers are supposed to contribute in order to receive a certain service in return. Some systems, for example *eMule*, are based on indirect reciprocity where peers can earn credits for their contributions. eMule is a light-weight, pair-wise credit system, and there are more complex payment-based incentive mechanisms such as the *Karma* system [2] or the now defunct *MojoNation*. However, these solutions typically require either a central administration or a distributed storage for the credits which is prone to cheating and attacks (e.g., Sybil attacks, whitewashing, etc., see also [3]). *KaZaA* is an example of an early system based on *direct reciprocity* that is rather easy to exploit: peers announce their "participation level" themselves; remote peers offer a prioritized service to participants that claim high contribution levels, without any verification. More recently, Menasché et al. [4] have compared the different types of reciprocity. The authors prove that under certain circumstances, direct reciprocity can emulate indirect reciprocity (i.e., credit-based systems). However, users must be willing to download undemanded content for bartering purposes and may use up to twice the bandwidth they would use under indirect reciprocity. The authors also describe a broker architecture (e.g., a sophisticated BitTorrent tracker) and study the system performance by simulation.

As mentioned before, the BitTorrent protocol is also based on direct reciprocity, but it is not a pure tit-for-tat system as peers also upload data "optimistically" for free, which can be exploited by selfish [5] or even free-riding clients [6]. Since the throughput of a strict tit-for-tat system is lower, such a system must employ certain methods to improve its performance, e.g., *source coding* [7], [8] or *network coding* [9]. Indeed, coding schemes constitute an alternative approach to increase the market liquidity of peer-to-peer systems: linear combinations of blocks are computed and distributed in place of the original blocks, yielding a higher block diversity and hence more trading opportunities. It is worth noting that coding schemes can easily be combined with inter-swarm and cyclic trading. The drawback of coding approaches, however, is their high computational complexity. A different approach is taken by Levin et al. [10] who assume an auction perspective on BitTorrent and study a modified BitTorrent protocol in which peers reward one another with proportional shares of bandwidth. Using game theory, the authors show that a proportional-share client is strategy-proof.

The idea of inter-swarm trading is not new. Guo et al. [11] argue that inter-swarm collaboration in BitTorrent is more effective than, e.g., directly stimulating seeders to stay in the network. They initiate the discussion of a multi-torrent system featuring a tracker site overlay and exchange-based trades along cycles. Aperjis et al. [12] adopt a more theoretical perspective and prove that bilateral equilibrium allocations are not Pareto-efficient in general, in contrast to multilateral allocations. Their work is related to a graph-theoretic generalization of classical *Arrow-Debreu* economics where edges in the graph indicate which entities can engage in direct trades [13]. The authors also provide quantitative insights (using BitTorrent data) into strategies where bilateral exchanges may perform quite well. Capotă et al. [14] formalize a resource (i.e., upload bandwidth) allocation problem in

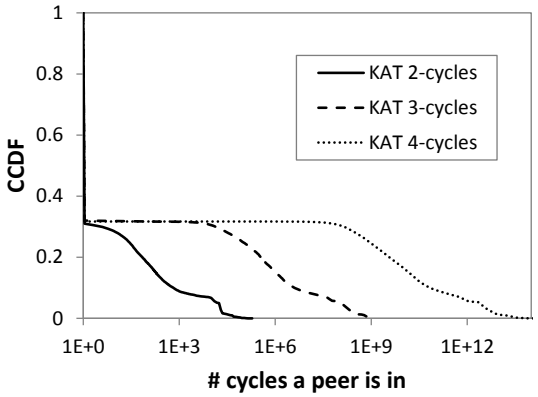

Fig. 4. Complementary cumulative distribution function of the number of inter-swarm cycles in which a peer resides; for all peers found in the top 1000 video torrents of KAT, with respect to cycles within these torrents.

BitTorrent communities across swarms and find that for file-sharing (goal: maximal throughput) and video-streaming (goal: serve as many users as possible with a stream of a certain quality) communities there is a high price of anarchy and that today's seeders' torrent selection mechanism is suboptimal: if peers uploaded (or seeded) in a different subset of the available torrents, the overall performance would be higher. Finally, Anagnostakis and Greenwald [15] discuss multi-lateral tit-for-tat trading from an incentive-compatibility perspective. Using simulations, the authors show that their proposed algorithm provides real incentives in the sense that free riders experience a poor service.

In contrast to the previous works reviewed above, we provide insights into the tradeoffs of cycle-based incentive mechanisms in different scenarios, focusing on the potential throughput gain as a function of the main parameters such as the cycle length. In addition, we describe a distributed algorithm to compute cycles and to avoid redundant downloads, and use realistic BitTorrent data traces to evaluate its performance.

III. Model

We consider a peer-to-peer system where peers interested in the same content are organized into *swarms*, i.e, in every swarm specific content is shared. This content is divided into multiple data *blocks*, and a block is the basic unit of trade. In practice, a peer typically joins a swarm by contacting a so-called *tracker*, a system entity whose main objective is to keep track of the peers in the swarm. Once a new peer informs the tracker that it wants to join, the tracker stores this peer's address and returns a list of addresses of other peers in the swarm. Instead of periodically querying the tracker, the peers can discover additional peers by exchanging addresses amongst themselves. In the first part of the paper, we make the simplifying assumption that peers are fully connected inside a swarm. The effect of trading with only a small subset of all possible peers is discussed in Section V-E.

Formally, we are given a set $\mathcal{P}$ of peers and a set $\mathcal{S}$ of swarms. Each peer $p \in \mathcal{P}$ joins a subset $\mathcal{S}_p \subseteq \mathcal{S}$ of swarms over time, where $\mathcal{S}_p$ is the set of swarms whose content peer p is interested in. We assume that a peer has a certain *upload* and *download bandwidth* available. While a peer p has incomplete downloads it tries to acquire file blocks by direct or indirect trades with peers that have interesting blocks to offer. A snapshot of the current peer interests can be modeled using a (dynamic) directed graph, which we call the *demand graph*: The node set is $\mathcal{P}$, and there is a directed edge from peer p_1 to peer p_2 if p_1 is interested in at least one block offered by peer p_2 in some swarm. Each peer can obtain a local approximate view of the demand graph by communicating with other peers and thus trade blocks along interest cycles of various length (see Section IV).

While a peer is in the process of downloading a certain content, it is called *leecher*, whereas a peer that has already obtained all blocks is called *seeder* in the respective swarm. Naturally, a peer participating in multiple swarms can be a seeder in some swarms and a leecher in others at the same time. Once a peer p has acquired the content of all swarms in $\mathcal{S}_p$, it leaves the system. Note that before leaving the system, it is in the peer's interest to stay in a swarm even after it has downloaded the corresponding content because there might still be opportunities to provide blocks from this swarm in exchange for blocks traded in other swarms. Thus, in a game-theoretic sense, seeding is a rational behavior. Compared to the standard intra-swarm trading, this incentive for peers to stay connected after they become seeders is a remarkable advantage of trading along cycles.

IV. Algorithm

In this section, we introduce the core algorithmic aspects, that is, we discuss how a local approximation of the demand graph is computed and particularly how cycles are selected for trading. As it is too costly to maintain an approximation of the entire demand graph, each peer p only keeps track of its k-neighborhood for some constant $k \geq 1$, which is the set of peers that can be reached from p with at most k hops in the demand graph. This is achieved by regularly exchanging peer lists and information about locally available blocks with all peers in the k-neighborhood and the trackers. Naturally, the peers only know their immediate neighbors in the beginning, but they quickly get to know their k-neighborhood by communicating with the peers in their neighbors' peer lists. Given the k-neighborhood, a peer computes possible trading cycles by exploring the demand graph. Since we only consider cycles of short length, a brute-force approach is feasible to find cycles in the demand graph; however, heuristics may be used to prune the search space. The process of computing trading cycles is explained in more detail in Section VI.

The *trading policy* defines which cycles are used to share data blocks and how the upload capacity is allocated. In order to capture the direct effect of trading along cycles, we investigate the following straightforward policy:

`Cycle(k)`: The peer participates in *any* trading cycle that is of length at most k. The bandwidth is allocated equally among all active cycles. If the data flow on a cycle is

diminished due to constraints by other peers, the unused bandwidth is allocated evenly among the remaining cycles.

Blocks are traded in a tit-for-tat-like manner on each cycle by having each peer p in the cycle send one block after the other to the peer that is interested in p's blocks, i.e., the blocks move opposite to the direction of the edges in the corresponding demand graph. In order to ensure fair trading, each peer maintains a balance between the number of uploaded and downloaded blocks for each active cycle. If the number of uploaded blocks is some constant τ larger than the number of downloaded blocks, the peer waits until the imbalance becomes smaller before sending out the next block. For our evaluation, we look at the most restrictive case of $\tau := 1$. Once a peer loses interest in its trading partner in a cycle, the corresponding edge in the demand graph vanishes and trading on this cycle ceases. Whenever a peer wants to upload a certain block, the block is put into a FIFO queue, i.e., the blocks are sent sequentially, and the actual upload time depends on the available bandwidth at the peer. All peers use a *uniform* block selection strategy, i.e., when requesting a block from a neighbor a peer selects a random block out of the interesting blocks that are requestable and non-pending. If no such block is available the peer re-requests a random pending block, i.e., a block that has been requested but not yet received. The impact of this trading strategy on the system performance is evaluated in the following section.

V. Evaluation

The main objective is to evaluate the influence of the parameter k on the market liquidity, i.e., the goal is to quantify the impact on the overall throughput. Naturally, a larger k yields more potential trading cycles; on the other hand, the message and computational complexity grows rapidly with increasing k. Therefore, it is of paramount importance to choose the right value for k, taking the overhead into account. Apart from studying the influence of k, we also compare `Cycle(`k`)` to the `Intra-swarm` policy where only bilateral, intra-swarm exchanges are allowed, and the bandwidth is also allocated equally among all active trades. This comparison reveals the potential gain of trading on cycles. Furthermore, as the overhead of `Cycle(`k`)` may become quite large, simply because *all* cycles are considered, we propose refinements that significantly increase the efficiency.

A. Methodology

There are two options to evaluate the different trading policies: simulating a trading system or conducting extensive measurements in a real distributed system. We opt for a simulation on packet level and emulate peers that run the distributed algorithm. One major advantage of a simulation approach is that we can ensure the same conditions (e.g., network congestion, CPU and memory usage) in all executions, which enables a fair comparison of the various policies. Moreover, a simulation test run takes considerably less time than a real time experiment.[3] These desirable properties would be lost when conducting measurements, e.g., on a testbed such as PlanetLab. A more insightful approach would be to have a real client that is actively used on the public Internet to trade files; however, this is not a promising approach as it depends on how well the client is received and how willing people are to share information about their downloads. For these reasons, we believe that the benefits of running a proper simulation, in particular the ability to study the effect of a parameter change in isolation, easily outweighs the shortcomings, which is mainly the loss of some network properties.

B. Simulation

We implemented an event-driven simulator that allows us to create multiple swarms where peers can join and leave over time. The simulator models the execution on a packet level, and both the size of the file shared in a swarm as well as the block size are parameterized. By default, we use 512 MB files and blocks of size 512 kB.

For simplicity, we do not consider any bounds on the download bandwidth in our evaluation. However, we limit the *upload bandwidth* of each peer to 500 kB/s. In order to inject data into the system, we assume the presence of a designated *seeder* in each swarm that provides any leecher with free file blocks at a constant rate of 10 kB/s.[4] These publishers do not engage in any trading otherwise. Regarding latencies, we use two models: one with constant latencies of 60ms between all peers, and one where the latencies capture the distribution of peers over three continents (e.g., Asia, Europe, America), that is, we draw the latency of a connection from three different Gauss distributions depending on the number of continental hops (i.e., $\mathcal{N}(30, 10)$ for transmissions within a continent, $\mathcal{N}(60, 10)$ for one continental hop, and $\mathcal{N}(90, 10)$ for two continental hops). Since both latency models yield similar results, we will only present the results for constant latencies in the following.

To model practical systems, we use a snapshot of the BitTorrent economy gathered by Zhang et al. [1]. Since such a snapshot contains millions of nodes and even more interactions, it is not feasible to use the entire trace in the simulation. This is not critical as the interactions change anyway when a different trading policy is used. We use the trace to determine the cardinality $D_p = |\mathcal{S}_p|$ for each peer p in our evaluation. In particular, we compute the number of downloads for all peers in the data set and store these values in a data set $\mathcal{D}$. The total number of peers and swarms is determined using the same BitTorrent snapshot: Since there are 3.65 times more peers than swarms in the data set, we simulate 365 peers and 100 swarms in the first part of the evaluation.

[3] In order to get the results presented in this paper, we simulated a total of more than a thousand test runs. A similar amount of test runs would take more than 20 months in real time.

[4] Existing systems typically rely on peers willing to provide blocks for free in order to solve the bootstrap problem. While some of these providers might have altruistic motives, others might have an interest in the actual dissemination of the content.

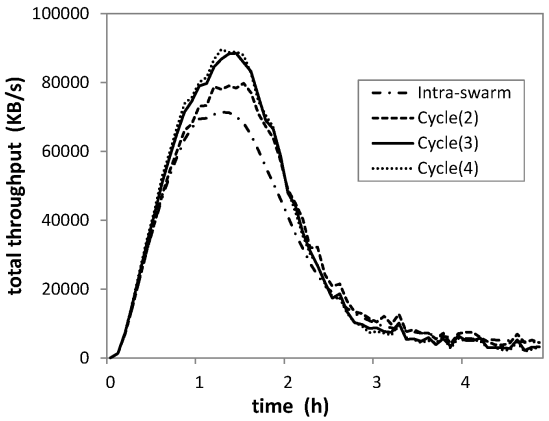

Fig. 5. Total throughput over time.

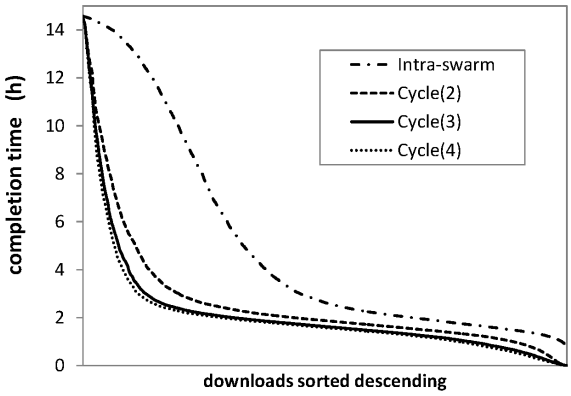

Fig. 6. Distribution of download completion times.

A single run of the simulation proceeds as follows. The simulations start at time $t = 0$ with $n = 100$ swarms each containing only one designated publishing seeder. Each peer $p \in \mathcal{P}$ is assigned the total number D_p of downloads it will start during the simulation by randomly choosing a value from the set $\mathcal{D}$. The D_p swarms for each peer p are chosen uniformly at random from all swarms $\mathcal{S}$. Since a peer typically joins its swarms over time, we model the time when a peer p joins the next swarm in $\mathcal{S}_p$ using a Poisson process with parameter $\lambda = 10^{-1}\text{min}^{-1}$ until it has joined all D_p swarms, i.e., we assume that the intervals between join events follow an exponential distribution. Whenever a new download starts, a peer joins the corresponding swarm as a leecher. Recall that a peer never leaves a swarm until *all* its downloads are completed.

We conducted several simulations where all peers use the policy `Intra-swarm`, `Cycle`(2), `Cycle`(3), or `Cycle`(4). Note that although both `Intra-swarm` and `Cycle`(2) restrict the peers to bilateral exchanges, they differ in that the latter allows for inter-swarm exchanges. Each scenario is executed with ten different sample sets and the results are averaged.

C. First Results

We start by analyzing the total throughput of all peers over time. Figure 5 depicts the averaged throughputs for the four different trading policies. The results confirm our expectation that `Intra-swarm` achieves the lowest throughput because it is the most restrictive trading policy, and the throughput increases when larger trading cycles are used. Interestingly, using `Cycle`(2) already results in a significant improvement: The peak download rate increases by roughly 12%. The policy `Cycle`(3) achieves even better results (24%); however, `Cycle`(4) only slightly outperforms `Cycle`(3) (26%).

The distribution of the download completion times is even more insightful. Figure 6 shows the completion times of all downloads sorted in descending order. It is clearly visible that the download rates improve substantially when inter-swarm trading is allowed. On average, a download completes within 5 hours 4 minutes with `Intra-swarm`, 2 hours 35 minutes with `Cycle`(2), 2 hours 8 minutes with `Cycle`(3), and 2 hours 1 minute with `Cycle`(4). Thus, the average download completion times are reduced by 49% when allowing trades on 2-cycles, and by 58% when additionally using 3-cycles. The improvement in terms of the median download completion times is 28% for `Cycle`(2) and 37% for `Cycle`(3). Again, the difference between `Cycle`(3) and `Cycle`(4) is negligible. We also recorded the completion times of all downloads in each individual simulation run: Compared to `Intra-swarm`, 84.4% of all downloads complete faster with `Cycle`(2), and the median improvement is 8.2%. When `Cycle`(3) is used, 96.9% of all downloads have a smaller completion time, and the median improvement is 14.7%. The same numbers for `Cycle`(4) are 97.4% and 16.1%, i.e., the numbers for `Cycle`(3) and `Cycle`(4) are again quite similar.

The investigation of completion times shows that while cyclic trading does not increase the peak throughput of the system tremendously (12% for 2-cycles and 24% for 3-cycles), the average download completion times, and therefore also the average download rate per download, improve significantly. One of the reasons is that cyclic trading especially helps at the beginning of a download, when the peers do not need to get initial blocks from seeders only, as well as towards the end of a download when the final missing blocks are collected more efficiently thanks to inter-swarm trades. Additionally, Figure 6 shows that more downloads have a completion time close to the median when trading on cycles, i.e., the resource allocation is more balanced. Note that this also indicates a higher degree of fairness when trading on cycles.

D. Avoiding Redundancy

From our first experiments we can conclude that the increased liquidity due to additional trading opportunities along cycles indeed leads to an increase of the overall performance. However, there is an issue that needs to be addressed: The same blocks can be requested in different cycles, which results in redundant downloads. The reason is that, as defined in Section IV, a peer requests a *pending block* if there is no available block that has not been requested already. This behavior is reasonable to some extent as pending requests might remain unanswered due to network failures, or slow

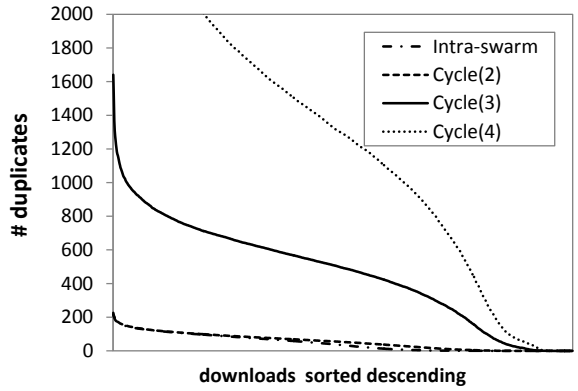

Fig. 7. Distribution of duplicates per download for each trading policy in descending order.

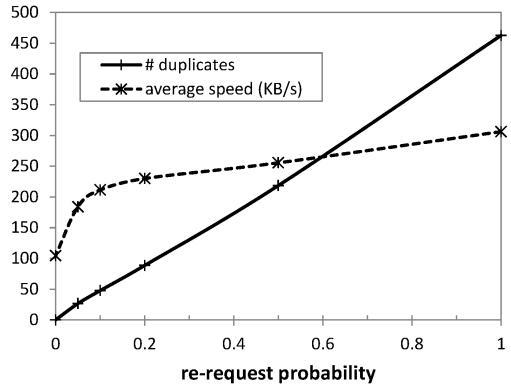

Fig. 8. Tradeoff between average number of duplicates and the average download rate with varying re-request probabilities for `Cycle(3)`.

connections might be worth replacing with faster connections. Moreover, it guarantees that all members of a cycle are always willing to trade on the cycle as long as the cycle exists in the demand graph. Unfortunately, such a nonrestrictive policy on re-requests leads to an intolerable number of redundant blocks as soon as the peers trade on cycles longer than 2. Figure 7 shows that peers download up to 1500 redundant blocks for a download consisting of 1024 blocks when using `Cycle(3)`, and up to 2000 blocks when using `Cycle(4)`.

The redundancy problem arises because the number of trading cycles that a neighboring peer p appears in is often larger than the number of interesting blocks that p has to offer. As a countermeasure, we propose the following two modifications to the trading policy `Cycle`(k).

1) **Selecting Cycles:** Limit the number of active cycles per neighbor proportionally to the number of blocks it can provide.
2) **Probabilistic Re-Request:** Re-request pending blocks only with a certain probability ρ.

Figure 8 depicts the tradeoff between probabilistic re-requests and the throughput: While the number of duplicates depends linearly on the re-request probability, the throughput grows quickly as long as ρ is fairly small. Thus, a smart re-request strategy can significantly reduce the number of duplicates without severely impacting the performance of the system.

In comparison with the measure of limiting the number of cycles that are selected for trading, probabilistic re-requests prove to be quite effective: while the use of Selection with `Cycle(3)` reduces the average number of duplicates from 462 to 266, probabilistic re-request with parameter $\rho = 0.1$ achieves a reduction to 48 duplicates on average, i.e., a reduction by almost 90%. It is evident that using probabilistic re-requests reduces the number of duplicates much more than limiting the selection of cycles. However, since we would like a systems that incurs a redundancy of less than 5% for the majority of downloads, we combine the two measures in the following to achieve an average redundancy of 27 duplicates per download in the case of `Cycle(3)`. Depending on the maximum cycle length used in a trading policy, we adapt the re-request probability ρ to achieve a good tradeoff between redundancy and download rate. In particular, we use a re-request probability of $\rho = 0.5$ for `Intra-swarm` and `Cycle(2)`, and $\rho = 0.1$ for `Cycle(3)` and `Cycle(4)`.

Another natural strategy to mitigate the redundancy problem would be to sort the cycles according to their "capacity", which we define as the number of blocks that could be traded along the cycle before the first peer loses interest and the cycle breaks. However, in our experiments, the performance gains were almost negligible and therefore not worth the additional algorithmic complexity.

In the following, we reassess the throughput benefits of our general block trading algorithms with our two anti-redundancy mechanisms in place. Regarding redundancy, probabilistic re-requesting and limiting the number of active cycles ensured that the median number of duplicates per download is 30 (i.e., an overhead of 2.9%) or less for all policies simulated; the 99^{th} percentile is below 90 (8.8%), and the maximum number of duplicates is over 90 only for the `Cycle(2)` policy.

Regarding performance, the adapted simulations yield median download completion times of 3 hours 59 minutes for `Intra-swarm`, 1 hour 57 minutes for `Cycle(2)`, 1 hour 41 minutes for `Cycle(3)`, and 1 hour 39 minutes for `Cycle(4)`. The average download completion time is 5 hours 43 minutes with `Intra-swarm`, 3 hours with `Cycle(2)`, 2 hours 50 minutes with `Cycle(3)`, and 2 hours 39 minutes for `Cycle(4)`. Qualitatively, the results are very similar to the results without anti-redundancy measures in terms of average download completion times: `Cycle(2)` and `Cycle(3)` perform 47% and 51% better than `Intra-swarm`, respectively. The improvement in terms of the median completion times is even higher, i.e., 51% for `Cycle(2)` and 58% for `Cycle(3)`. Compared to the simulations without any redundancy-reducing measures, the loss in performance is small, e.g., the median download completion time is increased by less than 5% for all cyclic trading policies.

Finally, we would like to point out the interesting issue of *coordination* that occurs as soon as peers are willing to trade only on a subset of all the cycles they are part of. If a peer decides to trade on only a few of potentially thousands

of cycles present, it can happen that there is at least one peer unwilling to trade on every cycle. Thus, the performance degrades simply because the peers do not agree on *which* cycles to trade. This phenomenon is naturally more prevalent the more cycles there are to chose from, i.e., for larger k. The lack of coordination is also the reason for the fact that the overall average download rate is lower for `Cycle`(4) than for `Cycle`(3). Fortunately, this issue does not affect performance significantly if the peers use only cycles up to length 3 since still more than two thirds of all negotiations succeed.

E. Active Set Trading Policy

So far, we have assumed that the peers in a swarm are all connected to each other. While this is possible to a certain extent, the cost of interacting with too many neighbors can be large (especially if TCP connections are used). In the BitTorrent protocol, peers only trade actively with a small subset of neighbors, the so-called *active set*. A similar extension is also possible in our scenario. We have conducted experiments where peers trade with a small number of neighbors, in particular, we allow each peer to connect to only 10 peers per swarm. As a result, a peer can learn only about a subgraph of the demand graph, and trade only on cycles involving neighbors in the active set. Interesting peers are always added to the active set if the set is not full. Each peer periodically assesses the active peers in terms of the amount of data received since the last assessment and replaces the worst peer by another random peer in the swarm. Moreover, peers are replaced immediately when they become uninteresting.

Figure 9 depicts the download completion time distributions for different policies and their counterparts using the redundancy-reducing measures and active sets. In our simulations, active sets containing at most 10 peers lead to a throughput decrease of at most 5%, and combining active sets with redundancy-reducing measures results in a total loss of at most 10%. Since this performance decrease is modest, the refined `Cycle`(k) policies still greatly outperform the basic `Intra-swarm` policy.

Introducing active sets does not increase the number of duplicates downloaded: all policies exhibit a median redundancy of less than 3%, i.e., the median number of redundantly downloaded blocks over all downloads is 30, and the maximum redundancy is less than 10%. Furthermore, the *overhead* of building up the local view of the demand graph is small as only little data needs to be transferred. In our simulations, we start a breadth first search along the edges up to k hops in the demand graph whenever a new edge appears. We account for the communication overhead incurred by a peer p by aggregating the number of times an outgoing edge of p was traversed in all search processes, and multiply it by the number of bits needed to represent a respective search message.

In all simulation runs, the maximum overhead per peer was below 0.8% of the content size downloaded by the peer, except for the scenario where peers employ `Cycle`(4). The overhead for `Cycle`(4) without redundancy-reducing measures is slightly larger at up to 2%, with active sets and redundancy-reducing measures, it grows to an intolerable level of 22%.

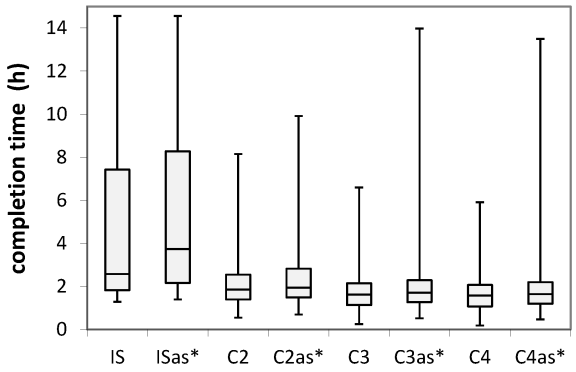

Fig. 9. Comparison of the download completion time distributions of the `Intra-swarm` (IS) policy and the `Cycle`(k) policies for $k = 2, 3, 4$ (C2, C3, C4) and their versions with active sets of size 10 as well as redundancy-reducing measures in place (ISas*, C2as*, C3as*, C4as*). The three horizontal lines of each box in a box plot depict the lower quartile, the median, and the upper quartile, and the end of the two whiskers represent the 5^{th} and the 95^{th} percentile.

As a conclusion of our simulations with $|\mathcal{P}| = 365$ and $|\mathcal{S}| = 100$ we propose to use `Cycle`(3) with active sets, probabilistic re-request, and the cycle selection measure to achieve an improvement of more than one third in terms of median download completion time, and the average download completes 44% faster. This is compared to `Intra-swarm`, which exhibits 5% redundancy on average, whereas the overhead due to redundant downloads of the proposed method amounts to only 2.6%. Moreover, using this method shortens the completion time in more than 93% of all downloads.

As a simpler alternative, one might also use `Cycle`(2) with the respective extensions to achieve an improvement of the download completion times of 25% (median) and 42% (average) with even less overhead and redundancy.

F. Modeling Preferences

In this section, we propose and evaluate a more sophisticated and arguably more realistic model for the peer preferences. So far, we have assumed a uniform model of peer preferences, where the peers choose the swarms they will join uniformly at random from the swarm set $\mathcal{S}$. This simple model ignores the fact that the users of file sharing systems typically have specific interests, which implies more clustered mappings between peers and swarms. We study the *clustering coefficient* distribution in the undirected graph consisting of the node set $\mathcal{P}$ and edges between peers that appear in at least one common swarm. The clustering coefficient of a peer is the number of edges between neighboring peers in this graph divided by the maximum number of such edges.

The uniform choice of swarms produces a demand graph whose clustering coefficients are too low if the number of peers and swarms is sufficiently large. In order to demonstrate this, we compare the clustering coefficients produced by our simulator with uniform preferences to the clustering coefficients we computed from the BitTorrent snapshot. As Figure 10 illustrates, if we keep the ratio of $n = |\mathcal{P}|$ and $|\mathcal{S}|$ constant at 3.65 the clustering coefficients decrease with growing n.

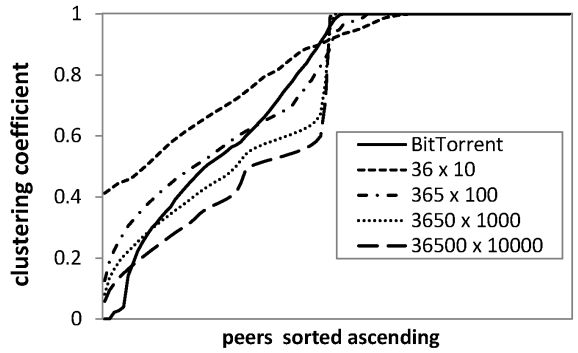

Fig. 10. Clustering coefficients with uniform preferences for different scales $|\mathcal{P}| \times |\mathcal{S}|$.

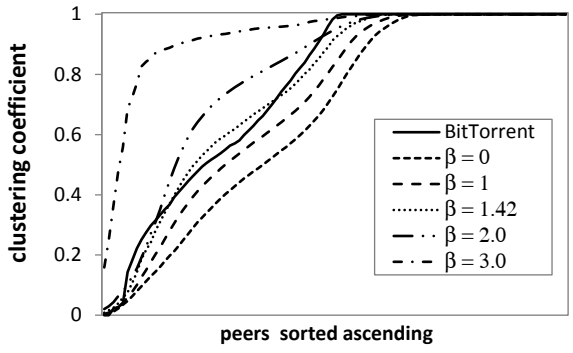

Fig. 11. Clustering coefficient distributions for $|\mathcal{P}| = |\mathcal{S}| = 100$, $\alpha = 1$ and varying β. The error is minimized for $\beta \approx 1.42$.

For $n = 36$ almost all clustering coefficients are too high, whereas for $n = 3650$ the clustering coefficient of most peers with more than one download is significantly too low. The peers with clustering coefficients of 1 are mostly peers with only one download. The explanation is that with more swarms available the probability that two peers in one swarm also appear together in another swarm decreases although the average swarm size remains constant.

Consequently, to reflect user preferences and thus the true correlation between peers and swarms more precisely, we propose a preference model that combines the concepts of *preferential attachment*: a popular swarm is likely to become more popular in the future, and *co-occurrence*: if a peer already shares many other interests with the peers in a given swarm S, it is more likely to join S as well (for a motivation see, e.g., [16]). Whenever a peer p starts a new download it joins swarm S with a probability proportional to the number of p's neighbors in S (co-occurrence) and the size of S (preferential attachment), i.e.,

$$Pr[p \text{ enters } S] \approx \frac{(|N_p \cap S| + 1)^\alpha + (|S| + 1)^\beta}{\sum_{X \in \mathcal{S}} (|N_p \cap X| + 1)^\alpha + (|X| + 1)^\beta}$$

where N_p denotes the set of p's neighbors. The parameters α and β allow us to model arbitrary combinations of the two concepts. For $\alpha = 0$, our preference model is a pure preferential attachment model, and for $\beta = 0$ it is a pure co-occurrence model. For $\alpha = \beta = 0$, we get a uniform distribution.

Given the proposed model, we can set up simulations that approximate both the clustering coefficient distribution and the swarm size distribution well by using a peer-swarm ratio of 3.65 and fit the clustering coefficient distribution by adapting α and β. However, as larger α and β values lead to larger clustering coefficients, we must consider a scenario where the uniform distribution results in clustering coefficients that are too low. As Figure 10 illustrates, this is the case for large peer and swarm sets when keeping the peer-swarm ratio constant. Since the computational complexity of the simulation increases quickly with the number of peers and swarms—and we also want to be able to fit the parameters for small networks—, the best option is to abandon the requirement to keep the peer-ratio set to 3.65. We found that by using $|\mathcal{P}| = |\mathcal{S}| = 100$ the swarms contain fewer peers on average than in the BitTorrent data; however, the clustering coefficients with uniform preferences are well below the BitTorrent clustering coefficients. Figure 11 depicts the clustering coefficient distribution observed in BitTorrent, and the clustering coefficient distributions produced by our simulator for various β.

We fitted the parameters α and β of the co-occurrence preference model using the method of least squares, i.e., α and β are set to values that minimize the error $\frac{1}{n}\sum_{i=1}^{n}(c_i - \hat{c}_i)^2$, where $\hat{c}_i$ is the i-th lowest clustering coefficient in the simulation, and c_i is the clustering coefficient at the corresponding position in the measured clustering coefficient distribution. Our study indicates that the choice of β has a large impact on the clustering coefficient distribution (cf. Figure 11), whereas the choice of α does not impact the clustering coefficients much for the relatively small numbers of peers and swarms used in our setup. The parameter α has a slight effect on the steepness of the curve.[5] As the error is similar for any $\alpha < 5$ by selecting the best value for β, we chose $\alpha = 1$ for simplicity. The respective optimal choice for β is 1.42, yielding an error smaller than 10^{-3}.

Figure 12 shows the download completion times in the scenario with 100 peers and 100 swarms using $\alpha = 1$ and $\beta = 1.42$. The results are similar to those in earlier experiments, and we can draw the same conclusions: `Cycle`(k) clearly outperforms `Intra-swarm` for any $k = 2, 3, 4$. Again, the figure shows that while the throughput rises with increasing k, the difference between `Cycle`(3) and `Cycle`(4) is marginal. We conjecture that the results also hold for a larger number of peers and swarms.

VI. Distributed Implementation

Having presented our evaluation results, we now discuss the distributed protocol, CycT4T, in order to fill in the blanks in the high-level description of the algorithm in Section IV. In particular, CycT4T enables peers to find cycles and to negotiate on which cycles to trade in an efficient manner. Note that CycT4T is the algorithm used in our simulation with the

[5]We believe the effect of co-occurrence will have a greater impact in larger systems.

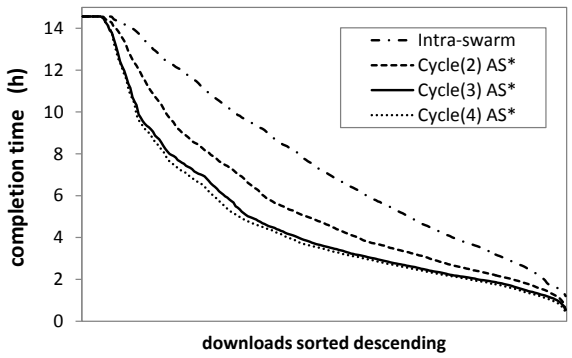

Fig. 12. Distribution of download completion times for $|\mathcal{P}| = |\mathcal{S}| = 100$, $\alpha = 1$, and $\beta = 1.42$.

exception that we merged the local views of the demand graph into a global view to save space.

CYCT4T uses the usual mechanisms like tracker polling, peer exchange (PEX), or distributed hash tables (DHT) to discover peers. Initially, two peers exchange information about their locally available blocks. After that, the peers only notify each other when new blocks become available.

In order to keep a local view of the demand graph, each peer p maintains two tables. The `outTable`, which contains all (known) interesting peers (*out-neighbors*), i.e., the peers possessing blocks that p is interested in, and the `inTable` whose purpose is to keep track of all peers from which there is a directed path of length at most $k-1$ in the demand graph ending at peer p. In particular, for each direct *in-neighbor* r, `inTable` stores the information *which* peers can reach p on paths P, $|P| < k$, where r is the last intermediate hop. The entries in the `inTable` consist of three values: *source*, which is the identifier of the peer at the beginning of the path, *via*, the identifier of the last peer on the path before p itself, and *distance*, the minimum length of all such paths. When p computes an update message for its out-neighbors the distance information is used to determine which `inTable`-entries are irrelevant. Peers also store direct in-neighbors r in the `inTable` as an entry $(r, r, 1)$. Note that if a peer q is in p's `outTable` and its `inTable` contains the entry $(q, r, d-1)$ then there exists at least one cycle $p \to q \to \ldots \to r \to p$ of length d for peers p, q, and r. Thus, the `inTable` and the `outTable` together allow p to decide whether it is in a cycle of length at most k with r as its predecessor and q as its successor for two given peers r,q.

Naturally, these tables must be built up initially and updated when there is a change in the vicinity. For this purpose, each peer informs the peers in its `outTable` about changes in its tables as follows. If peer p learns about an interesting peer q, it adds q to its `outTable` and sends q an *update list* of all relevant (source,distance)-pairs extracted from its own `inTable` (cf. Table I). Upon receiving an update list from a peer r, peer p increases all distances by 1, since one hop is added to the paths, and updates the corresponding entries with `via` $= r$ in its `inTable`. Afterwards, it recursively computes relevant (source,distance)-pairs and forwards them

UPD:	`SELECT DISTINCT source, distance FROM inTable WHERE distance` $< k-1$ `AND via` $\neq q$`;`
CID:	`SELECT source FROM outTable INNER JOIN inTable ON outTable.id = inTable.source WHERE via` $= r$`;`

TABLE I
SQL QUERIES FOR A FULL UPDATE MESSAGE TO q (UPD), AND FOR FINDING ALL OUT-NEIGHBORS ON CYCLES WITH IN-NEIGHBOR r (CID).

to the peers in its `outTable`. The tables must also be updated if an edge from a peer r to peer p disappears. In this case, p removes all entries where `via` $= r$ from its `inTable` and sends a *remove list* containing the removed entries to the peers in its `outTable`. If a peer receives such a remove list, it updates its `inTable` accordingly and computes an update message. For the sake of brevity, we omit the details of these computations and the precise contents of the update message. Basically, an update message must announce all changes to the minimum shortest path distance over all in-neighbors. Thus, modifications to an entry in the `inTable` concerning source q where `via` $= r$ must be announced to the peers in the `outTable` only if this modification changed the minimum distance from q via *any* in-neighbor.

Whenever peer p_1 adds an interesting peer p_2 to its `outTable`, it checks whether there is a cycle containing edge (p_1, p_2). As described earlier, p_1 can only determine whether there are such cycles, but not how many there are and not which peers they contain in particular (except for cycles of length smaller equal 3). In order to find all cycles with edge (p_1, p_2), peer p_1 sends a *cycle ID (CID) message* containing $h_{p_1}(p_1||p_2)$ to p_2, where h_{p_1} is a hash function private to p_1 and $||$ denotes the concatenation operator. The hash functions h_p are required to produce hash values of publicly known constant length. Peer p_2 in turn determines the set X of out-neighbors that are part of a cycle to p_1 (cf. Table I), and it forwards the received CID with its private hash value $h_{p_2}(p_2||x)$ appended to each peer $x \in X$. The peers receiving a CID message execute the same steps unless the list contains k hash values already, in which case they do not forward the CID any further. Peer p_1 will finally get a CID message $h_{p_1}(p_1||p_2)||h_{p_2}(p_2||p_3)||\ldots||h_{p_\ell}(p_\ell||p_1)$, where $\ell \leq k$, for each cycle. Although peer p_1 cannot decrypt the CID message, it can compute the cycle's length and recognize the head $h_{p_1}(p_1||p_2)$. Furthermore, a unique cycle ID is computed by XORing all contained hash values, $h_{p_1}(p_1||p_2), h_{p_2}(p_2||p_3), \ldots, h_{p_\ell}(p_\ell||p_1)$. The cycle ID is appended to each future data message, indicating that this transfer is a contribution to the trade on this particular cycle. Note that the cycle ID is constructed such that each cycle is identified with its unique ID by every involved peer regardless of the order of the hash values. This prevents potential problems that could arise when a particular cycle is found by more than one search process. This can happen due to inconsistent approximations of the demand graph.

Since it is unknown in advance which and how many cycles

will be discovered, a second phase is required to select cycles for trading. For each potential cycle, peer p_1 initiates the negotiation by sending the hashes in the received CID message together with a negotiation bit, set to 1, to its out-neighbor in the cycle. Each peer in the cycle may set the negotiation bit to 0, indicating that it does not want to trade on this cycle, before it forwards the message. If the bit is still set to 1 when p_1 receives the message, this cycle is accepted for trading, otherwise it is discarded. In order to inform the other peers about the final decision, the result is sent around the cycle. Each peer starts trading as described in Section IV as soon as it learns that the negotiation was successful, i.e., it requests a desired block from the successor in the cycle. Of course, the peers do not start uploading a block at exactly the same time. However, this is not a critical issue as the tit-for-tat trading on cycles proposed in Section IV automatically mitigates temporal fluctuations and also differences of edge bandwidths: a peer with large upload bandwidth waits for the rest of the cycle to catch up as soon as the threshold on the local upload-download balance is reached for this cycle. Hence, the bandwidth expended at each peer for any given cycle tends towards the bandwidth of the slowest edge if the threshold is not too large.

VII. Conclusion

What are the potential gains in barter-based peer-to-peer systems when moving from bilateral trades to multi-lateral trades along cycles of interest? Our study shows that peer-to-peer systems can benefit from trading on cycles in the sense that the majority of peers obtains the desired content faster. Especially for short cycles, the throughput benefits are high and the overhead is low. We find that the best tradeoff is achieved using the `Cycle`(3) trading policy with active sets and probabilistic re-request. Also `Cycle`(2) can be very attractive due to its simplicity and low overhead.

The generalization of the basic tit-for-tat concept to trading on cycles has interesting game-theoretic implications. For example, it becomes rational for a peer to stay in a swarm after the download is complete, as the acquired content can still be used in cross-swarm trades. This is in stark contrast to bilateral intra-swarm bartering, where peers do not have an incentive to offer such content. Thus, the introduction of inter-swarm trades could motivate users to stay in swarms longer, thereby increasing the availability of content. It is worth noting that the generalization also opens up new ways of cheating for selfish peers. For example, if two peers *collude*, they can fake a multitude of 4-cycles with an imaginary fourth node to make a trustful node believe it resides in many 4-cycles and thus provides several free file blocks until the balance threshold is reached on all cycles. An easy way to counter such an attack is to keep an additional balance associated with peers rather than cycles: for every unreturned file block on a cycle with neighbors u and v decrease the balance (or the trust value) for both peers. If the balance is too low, no additional blocks are forwarded until new blocks are received. More work is needed to fully understand the economical aspects of trading on cycles: New forms of strategic behavior may hurt the system performance, e.g., a selfish peer may prefer shorter trading cycles because shorter cycles are easier to find, require less management overhead, and may be more stable. Moreover, while we discussed multiple optimizations and refinements of the basic trading policy, there are aspects of the protocol that might still be improved in future work, e.g., the bandwidth allocation to the different cycles. This can result in additional performance gains.

Acknowledgments. We are grateful to Chao Zhang, Prithula Dhungel, Di Wu and Keith W. Ross for providing us with the BitTorrent data. The authors would also like to thank Michael König, Ali Ghodsi, Bernhard Ager and Anja Feldmann for helping with the evaluation, and Klaus M. Schmidt from LMU Munich for interesting economical discussions.

References

[1] C. Zhang, P. Dhungel, D. Wu, and K. W. Ross, "Unraveling the BitTorrent Ecosystem," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 7, 2010.

[2] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, "KARMA: A Secure Economic Framework for Peer-to-Peer Resource Sharing," in *Proc. Workshop on Economics of Peer-to-Peer Systems*, 2003.

[3] R. Landa, D. Griffin, R. G. Clegg, E. Mykoniati, and M. Rio, "A Sybilproof Indirect Reciprocity Mechanism for Peer-to-Peer Networks," in *Proc. 28th IEEE International Conference on Computer Communications (INFOCOM)*, 2009.

[4] D. Menasché, L. Massoulié, and D. Towsley, "Reciprocity and Barter in Peer-to-Peer Systems," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, 2010.

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

[6] 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.

[7] D. Grolimund, L. Meisser, S. Schmid, and R. Wattenhofer, "Havelaar: A robust and efficient reputation system for active peer-to-peer systems," in *Proc. Workshop on the Economics of Networked Systems (NetEcon)*, pp. 69–74, 2006.

[8] T. Locher, S. Schmid, and R. Wattenhofer, "Rescuing Tit-for-Tat with Source Coding," in *7th IEEE International Conference on Peer-to-Peer Computing (P2P), Galway, Ireland*, September 2007.

[9] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network Information Flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204 –1216, 2000.

[10] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee, "Bittorrent is an auction: analyzing and improving bittorrent's incentives," in *Proc. ACM SIGCOMM*, pp. 243–254, 2008.

[11] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, Analysis, and Modeling of BitTorrent-like Systems," in *Proc. 5th ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2005.

[12] C. Aperjis, R. Johari, and M. J. Freedman, "Bilateral and Multilateral Exchanges for Peer-Assisted Content Distribution," in *Under submission*, 2011.

[13] S. M. Kakade, M. Kearns, and L. E. Ortiz, "Graphical Economics," in *Proc. 17th Annual Conference on Learning Theory (COLT)*, pp. 17–32, 2004.

[14] M. Capota, N. Andrade, T. Vinko, F. Santos, J. Pouwelse, and D. Epema, "Inter-swarm resource allocation in bittorrent communities," in *Proc. IEEE International Conference on Peer-to-Peer Computing (P2P)*, pp. 300 –309, 2011.

[15] K. Anagnostakis and M. Greenwald, "Exchange-based Incentive Mechanisms for Peer-to-Peer File Sharing," in *Proc. 24th International Conference on Distributed Computing Systems (ICDCS)*, 2004.

[16] G. Linden, B. Smith, and J. York, "Amazon.com Recommendations: Item-to-Item Collaborative Filtering," *Internet Computing, IEEE*, vol. 7, no. 1, pp. 76 – 80, 2003.