Taming Dynamic and Selfish Peers[*]

Fabian Kuhn[‡], Thomas Moscibroda[§], Stefan Schmid[§], Roger Wattenhofer[§]
kuhn@microsoft.com, moscitho@tik.ee.ethz.ch, schmiste@tik.ee.ethz.ch, wattenhofer@tik.ee.ethz.ch
[‡]Microsoft Research Silicon Valley, 1065 La Avenida, Mountain View, CA 94043, USA
[§]Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 8092 Zurich, Switzerland

Abstract

This paper addresses two important challenges for current P2P systems, namely churn and selfishness. First, we report on a system [19] whose desirable properties (small peer degree, small network diameter, etc.) are maintained in spite of ongoing and concurrent membership changes. Thereby, dynamic peers are "tamed" by redundancy. Due to the worst-case churn, this system may never be fully repaired, but always fully functional. However, it relies on the fact that peers act according to the protocol. In the second part of the paper (based on [22]), we study the impact of selfish peers which follow those protocols which maximize their utility. In particular, the efficiency of topologies formed by selfish peers is considered. We show that even in the absence of joins and leaves, the resulting system may never stabilize. How to "tame" selfish peers towards a more cooperative behavior remains an open issue.

1 CHURN

Many existing file sharing systems are faced with frequent membership changes ("churn"). While in Internet telephony applications such as Skype the users remain online for longer time periods, peers typically connect to a file sharing system only shortly (e.g., to download a small number of files).

Most P2P systems in the literature are only analyzed for static environments, or for scenarios where the system is given sufficient time to recover again after as set of peers has crashed. In contrast, we have developed algorithms to maintain desirable properties of a P2P topology in spite of ongoing and concurrent *worst-case* changes. We think of an adversary which can remove and add a bounded number of peers. The adversary cannot be fooled by any kind of randomness. It can choose which peers to crash and how peers join.[1] The adversary does not have to wait until the system is recovered before it crashes the next batch of peers. Instead, the adversary can constantly crash peers while the system is trying to stay alive. Indeed, our system is *never fully repaired* but *always fully functional*. In particular, our system is resilient against an adversary which continuously attacks the "weakest part" of the system. Our system counters such an adversary by continuously moving the remaining or newly joining peers towards the sparse areas.

Clearly, we cannot allow our adversary to have unbounded capabilities. In particular, in any constant time interval, the adversary can at most add and/or remove $O(\log n)$ peers, n being the total number of peers currently in the system. This model covers an adversary which repeatedly takes down machines by a distributed denial of service attack, however only

[*]Presented at Peer-to-Peer Systems and Applications Seminar, March 2006, at the International Conference and Research Center for Computer Science, Schloss Dagstuhl, Germany. Note that this paper is based on our work at IPTPS 2005 [19] and IPTPS 2006 [22]. Research is supported by the Swiss National Science Foundation and the Hasler Stiftung.

[1]We assume that a joining peer knows a peer which already belongs to the system. This is known as the *bootstrap* problem.

a logarithmic number of machines at each point in time. Our algorithm relies on messages being delivered timely, in at most constant time between any pair of operational peers. In distributed computing such a system is called *synchronous*. Note that if nodes are synchronized locally, our algorithm also runs in an asynchronous environment. In this case, the propagation delay of the slowest message defines the notion of time which is needed for the adversarial model.

The basic structure of our P2P system is a hypercube. Each peer is part of a distinct hypercube node; each hypercube node consists of $\Theta(\log n)$ peers. Peers have connections to other peers of their hypercube node and to peers of the neighboring hypercube nodes. In the case of joins or leaves, some of the peers have to change to another hypercube node such that up to constant factors, all hypercube nodes own the same number of peers at all times. If the total number of peers grows or shrinks above or below a certain threshold, the dimension of the hypercube is increased or decreased by one, respectively.

The balancing of peers among the hypercube nodes can be seen as a dynamic token distribution problem [23] on the hypercube. Each node of a graph (hypercube) has a certain number of tokens, the goal is to distribute the tokens along the edges of the graph such that all nodes end up with the same or almost the same number of tokens. While tokens are moved around, an adversary constantly inserts and deletes tokens. Our P2P system builds on two basic components: i) an algorithm which performs the described dynamic token distribution and ii) an information aggregation algorithm which is used to estimate the number of peers in the system and to adapt the dimension accordingly.

Based on the described structure, we get a fully scalable, efficient P2P system which tolerates $O(\log n)$ worst-case joins and/or crashes per constant time interval. As in other P2P systems, peers have $O(\log n)$ neighbors, and the usual operations (e.g. search) take time $O(\log n)$. In our view a main contribution of the paper, however, is to propose and study a model which allows for dynamic adversarial churn. We believe that our basic algorithms (dynamic token distribution and information aggregation) can be applied to other P2P topologies, such as butterflies, skip graphs, chordal rings, etc. It can even be used for P2P systems that go beyond distributed hash tables (DHT).

1.1 Related Work

A plethora of different overlay networks with various interesting technical properties have been proposed over the last years (e.g. [1, 4, 10, 11, 16, 18, 21, 26, 27, 32, 37]). Due to the nature of P2P systems, fault-tolerance has been a prime issue from the beginning. The systems usually tolerate a large number of random faults. However after crashing a few peers the systems are given sufficient time to recover again. From an experimental point of view, churn has been studied in [28], where practical design tradeoffs in the implementation of existing P2P networks are considered.

Resilience to worst-case failures has been studied by Fiat, Saia et al. in [15, 30]. They propose a system where, w.h.p., $(1-\varepsilon)$-fractions of peers and data survive the adversarial deletion of up to half of all nodes. In contrast to our work the failure model is static. Moreover, if the total number of peers changes by a constant factor, the whole structure has to be rebuilt from scratch.

Scalability and resilience to worst-case joins and leaves has been addressed by Abraham et al. in [3]. The focus lies on maintaining a balanced network rather than on fault-tolerance in the presence of concurrent faults. In contrast to our paper, whenever a join or leave happens, the network has some time to adapt.

The only paper which explicitly treats arbitrarily concurrent worst-case joins and leaves is by Li et al. [20]. In contrast to our work, Li et al. consider a completely asynchronous model where messages can be arbitrarily delayed. The stronger communication model is compensated by a weaker failure model. It is assumed that peers do not crash. Leaving peers execute an appropriate "exit" protocol and do not leave before the system allows this; crashes are not allowed.

1.2 Model

We consider the *synchronous message passing model*. In each round, each peer can send a message to all its neighbors. Additionally, we have an adversary $\mathcal{A}(J, L, \lambda)$ which may perform J arbitrary joins and and L arbitrary leaves (crashes) in each interval of λ rounds.

We assume that a joining peer π_1 contacts an arbitrary peer π_2 which already belongs to the system; π_2 then triggers the necessary actions for π_1's integration. A peer may be contacted by several joining peers simultaneously. In contrast to other systems where peers have to do some finalizing operations before leaving, we consider the more general case where peers depart or crash without notice.

1.3 Algorithm

In this section, we describe the maintenance algorithm which maintains the simulated hypercube in the presence of an adversary which constantly adds and removes peers. The goal of the maintenance algorithm is twofold. It guarantees that each node always contains at least one peer which stores the node's data. Further, it adapts the hypercube dimension to the total number of peers in the system.

This is achieved by two basic components. First, we present a dynamic token distribution algorithm for the hypercube. Second, we describe an information aggregation scheme which allows the nodes to simultaneously change the dimension of the hypercube.

1.3.1 Dynamic Token Distribution

The problem of distributing peers uniformly throughout a hypercube is a special instance of a *token distribution problem*, first introduced by Peleg and Upfal [23]. The problem has its origins in the area of load balancing, where the workload is modeled by a number of *tokens* or jobs of unit size; the main objective is to distribute the total load equally among the processors. Such load balancing problems arise in a number of parallel and distributed applications including job scheduling in operating systems, packet routing, large-scale differential equations and parallel finite element methods. More applications can be found in [31].

Formally, the goal of a token distribution algorithm is to minimize the maximum difference of tokens at any two nodes, denoted by the *discrepancy* ϕ. This problem has been studied intensively; however, most of the research is about the *static variant* of the problem, where given an arbitrary initial token distribution, the goal is to redistribute these tokens uniformly. In the *dynamic variant* on the other hand, the load is dynamic, that is, tokens may arrive and depart *during* the execution of the token distribution algorithm. In our case, peers may join and leave the simulated hypercube at arbitrary times, so the emphasis lies on the dynamic token distribution problem on a d-dimensional hypercube topology.

We use two variants of the token distribution problem: In the *fractional token distribution*, tokens are arbitrarily divisible, whereas in the *integer token distribution* tokens can only move as a whole. In our case, tokens represent peers and are inherently integer. However, it turns out that the study of the fractional model is useful for the analysis of the integer model.

We use a token distribution algorithm which is based on the *dimension exchange method* [13, 25]. Basically, the algorithm cycles continuously over the d dimensions of the hypercube. In step s, where $i = s \mod d$, every node $u := \beta_0 ... \beta_i ... \beta_{d-1}$ having a tokens balances its tokens with its adjacent node in dimension i, $v := \beta_0 ... \overline{\beta_i} ... \beta_{d-1}$, having b tokens, such that both nodes end up with $\frac{a+b}{2}$ tokens in the fractional token distribution. On the other hand, if the tokens are integer, one node is assigned $\lceil \frac{a+b}{2} \rceil$ tokens and the other one gets $\lfloor \frac{a+b}{2} \rfloor$ tokens.

It has been pointed out in [13] that the described algorithm yields a perfect discrepancy $\phi = 0$ after d steps for the static fractional token distribution. In [25], it has been shown that in the worst case, $\phi = d$ after d steps in the static integer token distribution. We can show that if the decision to which node to assign $\lceil \frac{a+b}{2} \rceil$ and to which node to assign $\lfloor \frac{a+b}{2} \rfloor$ tokens is made randomly, the final discrepancy is constant in expectation. However, we do not make use of this because it has no influence on our asymptotic results.

In the following, the dynamic integer token distribution problem is studied, where a "token adversary" $\mathcal{A}(J, L, 1)$ adds at most J and removes at most L tokens at the beginning of each step. In particular, we will show that if the initial distribution is perfect, i.e., $\phi = 0$, our algorithm maintains the invariant $\phi \leq 2J + 2L + d$ at every moment of time.

For the dynamic fractional token distribution, the tokens inserted and deleted at different times can be treated independently and be superposed. Therefore, the following lemma holds.

Lemma 1.1. *For the dynamic fractional token distribution, the number of tokens at a node depends only on the token insertions and deletions of the last d steps and on the total number of tokens in the system.*

We can now bound the discrepancy of the integer token distribution algorithm by comparing it with the fractional problem.

Lemma 1.2. *Let v be a node of the hypercube. Let $\tau_v(t)$ and $\tau_{v,f}(t)$ denote the number of tokens at v for the integer and fractional token distribution algorithms at time t, respectively. We have $\forall t$: $|\tau_v(t) - \tau_{v,f}(t)| \leq \frac{d}{2}$.*

Lemma 1.3. *In the presence of an adversary $\mathcal{A}(J, L, 1)$, it always holds that the integer discrepancy $\phi \leq 2J + 2L + d$.*

1.3.2 Information Aggregation

When the total number of peers in the d-dimensional hypercube system exceeds a certain threshold, all nodes $\beta_0 \ldots \beta_{d-1}$ have to split into two new nodes $\beta_0 \ldots \beta_{d-1}0$ and $\beta_0 \ldots \beta_{d-1}1$, yielding a $(d+1)$-dimensional hypercube. Analogously, if the number of peers falls beyond a certain threshold, nodes $\beta_0 \ldots \beta_{d-2}0$ and $\beta_0 \ldots \beta_{d-2}1$ have to merge their peers into a single node $\beta_0 \ldots \beta_{d-2}$, yielding a $(d-1)$-dimensional hypercube. Based on ideas also used in [7, 34, 33], we present an algorithm which provides the same estimated number of peers in the system to all nodes in every step allowing all nodes to split or merge synchronously, that is, in the same step. The description is again made in terms of *tokens* rather than peers.

Assume that in order to compute the total number of tokens in a d-dimensional hypercube, each node $v = \beta_0 ... \beta_{d-1}$ maintains an array $\Gamma_v[0...d]$, where $\Gamma_v[i]$ for $i \in [0, d]$ stores the estimated number of tokens in the sub-cube consisting of the nodes sharing v's prefix $\beta_0 ... \beta_{d-1-i}$. Further, assume that at the beginning of each step, an adversary inserts and removes an arbitrary number of tokens at arbitrary nodes. Each node $v = \beta_0 ... \beta_{d-1-i} ... \beta_{d-1}$ then calculates the new array $\Gamma'_v[0...d]$. For this, v sends $\Gamma_v[i]$ to its adjacent node $u = \beta_0 ... \overline{\beta_{d-1-i}} ... \beta_{d-1}$, for $i \in [0, d-1]$. Then, $\Gamma'_v[0]$ is set to the new number of tokens at v which is the only node with prefix $\beta_0 ... \beta_{d-1}$. For $i \in [1, d]$, the new estimated number of tokens in the prefix domain $\beta_0 ... \beta_{d-1-(i+1)}$ is given by the total number of tokens in the domain $\beta_0 ... \beta_{d-1-i}$ plus the total number of tokens in domain $\beta_0 ... \overline{\beta_{d-1-i}}$ provided by node u, that is, $\Gamma'_v[i+1] := \Gamma_v[i] + \Gamma_u[i]$.

Lemma 1.4. *Consider two arbitrary nodes v_1 and v_2 of the d-dimensional hypercube. Our algorithm guarantees that $\Gamma_{v_1}[d] = \Gamma_{v_2}[d]$ at all times t. Moreover, it holds that this value is the correct total number of tokens in the system at time $t - d$.*

1.4 Simulated Hypercube

Based on the components presented in the previous sections, both the topology and the maintenance algorithm are now described in detail. In particular, we show that, given an adversary $\mathcal{A}(d+1, d+1, 6)$

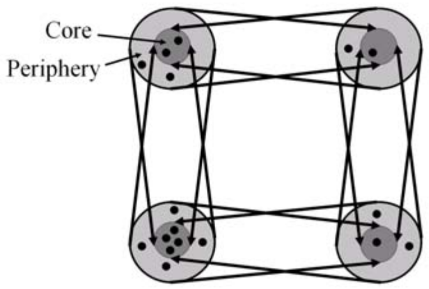

Figure 1: A simulated 2-dimensional hypercube with four nodes, each consisting of a core and a periphery. All peers within the same node are completely connected to each other, and additionally, all peers of a node are connected to all core peers of the neighboring nodes. Only the core peers store data items, while the peripheral peers may move between the nodes to balance biased adversarial changes.

which inserts and removes at most $d+1$ peers in any time interval of 6 rounds, 1) the out-degree of every peer is bounded by $\Theta(\log^2 n)$ where n is the total number of peers in the system, 2) the network diameter is bounded by $\Theta(\log n)$, and 3) every node of the simulated hypercube has always at least one peer which stores its data items, so no data item will ever be lost.

1.4.1 Topology

We start with a description of the overlay topology. As already mentioned, the peers are organized to simulate a d-dimensional hypercube, where the hypercube's nodes are represented by a group of peers. A data item with identifier id is stored at the node whose identifier matches the first d bits of the hash-value of id.

The peers of each node v are divided into a *core* $\mathcal{C}_v$ of at most $2d+3$ peers and a *periphery* $\mathcal{P}_v$ consisting of the remaining peers; all peers within the same node are completely connected (*intra-connections*). Moreover, every peer is connected to all *core* peers of the neighboring nodes (*inter-connections*). Figure 1 shows an example for $d=2$.

The data items belonging to node v are replicated on all core peers, while the peripheral peers are used for the balancing between the nodes according to the peer distribution algorithm and do not store any data items. The partition into core and periphery has the advantage that the peers which move between nodes do not have to replace the data of the old node by the data of the new nodes in most cases.

1.4.2 6-Round (Maintenance) Algorithm

The *6-round (maintenance) algorithm* maintains the simulated hypercube topology described in the previous section given an adversary $\mathcal{A}(d+1, d+1, 6)$. In particular, it ensures that 1) every node has at least one core peer all the times and hence no data is lost; 2) each node always has between $3d+10$ and $45d+86$ peers; 3) only peripheral peers are moved between nodes, thus the unnecessary copying of data is avoided.

In the following, we refer to a complete execution of all six rounds (round 1 – round 6) of the maintenance algorithm as a *phase*. Basically, the 6-round algorithm balances the peers across one dimension in every phase according to the token distribution algorithm as described in Section 1.3.1; additionally, the total number of peers in the system is computed with respect to an earlier state of the system by the information aggregation algorithm of Section 1.3.2 to expand or shrink the hypercube if the total number of peers exceeds or falls below a certain threshold. In our system, we use the lower threshold $LT := 8d+16$ and the upper threshold $UT := 40d+80$ for the total number of peers *per node on average*.[2]

While peers may join and leave the system at arbitrary times, the 6-round algorithm considers the (accumulated) changes only once per phase. That is, a snapshot of the system is made in round 1; rounds 2 – 6 then ignore the changes that might have happened in the meantime and depend solely on the snapshot

[2]Note that since we consider the threshold *on average*, and since these values are provided with a delay of d phases in a d-dimensional hypercube (see Lemma 1.4), the number of peers at an individual node may lie outside $[LT, UT]$.

at the beginning of the phase.

Round 1: Each node v makes the snapshot of the currently active peers. For this, each peer in v sends a packet with its own ID and the (potentially empty) ID set of its joiners to all adjacent peers *within* v.

Round 2: Based on the snapshot, the core peers of a node v know the total number of peers in the node and send this information to the neighboring core with which they have to balance in this phase (cf. Section 1.3.1). The cores also exchange the new estimated total number of peers in their domains with the corresponding adjacent cores (cf. Section 1.3.2). Finally, each peer informs its joiners about the snapshot.

Round 3: Given the snapshot, every peer within a node v can compute the new periphery (snapshot minus old core). This round also prepares the transfer for the peer distribution algorithm across dimension i: The smaller of the two nodes determines the peripheral peers that have to move and sends these IDs to the neighboring core.

Round 4: In this round, the peer distribution algorithm is continued: The core which received the IDs of the new peers sends this information to the periphery. Additionally, it informs the new peers about the neighboring cores, etc.

The dimension reduction is prepared if necessary: If the estimated total number of peers in the system is beyond the threshold, the core peers of a node which will be reduced send their data items plus the identifiers of all their peripheral peers (with respect to the situation *after* the transfer) to the core of their adjacent node in the largest dimension.

Round 5: This round finishes the peer distribution, establishes the new peripheries, and prepares the building of a new core. If the hypercube has to grow in this phase, the nodes start to split, and vice versa if the hypercube is going to shrink.

Given the number of transferred peers, all peers can now compute the new peripheries. Moreover, they can compute the new core: It consists of the peers of the old core which have still been alive in Round 1, plus the $2d + 3 - |\mathcal{C}|$ smallest IDs in the new periphery, where $\mathcal{C}$ is the set of the old core peers which have still been alive in Round 1. The old core then informs all its neighboring nodes (i.e., their old cores) about the new core.

If the hypercube has to grow in this phase, the smallest $2d + 3$ peers in the new periphery of the node that has to be split become the new core of the expanded node, and half of the remaining peripheral peers build its periphery. Moreover, the necessary data items are sent to the core of the expanded node, and the neighboring (old) cores are informed about the IDs of the expanded core.

If the hypercube is about to shrink, all old cores in the lower half of the hypercube (the surviving subcube) inform their periphery about the peers arriving from the expanded node and the peers in the expanded node about the new core and its periphery. The data items are copied to the peers as necessary.

Round 6: In this round, the new cores are finally built: The old core forwards the information about the new neighboring cores to the peers joining the core.

Moreover, if the hypercube has been reduced, every peer can now compute the new periphery. If the hypercube has grown, the old core forwards the expanded cores of its neighbors to *all* peers in its expanded node.

Theorem 1.5. *Given an adversary $\mathcal{A}(d+1, d+1, 6)$ which inserts and removes at most $d + 1$ peers per phase, the described 6-round algorithm ensures that 1) every node always has at least one core peer and hence no data is lost; 2) each node has between $3d + 10$ and $45d + 86$ peers, yielding a logarithmic network diameter; 3) only peripheral peers are moved between nodes, thus the unnecessary copying of data is avoided.*

In order to enhance clarity, we described a scheme which is as simple as possible. Instead of a complete bipartite graph between adjacent hypercube nodes one could e.g. use a bipartite matching. This reduces the node degree from $O(\log^2 n)$ to $O(\log n)$. Apart from better node degrees, all our results still hold up to constant factors.

1.5 Conclusions

We have reported on our approach to cope with churn in a P2P system. To the best of our knowledge, this is the first proposal which is able to cope with ongoing worst-case churn. Our techniques are generic and can be applied to many other topologies: We simply need a token distribution and an information aggregation algorithm on the corresponding graph.

However, of course, our system is only a first step. We believe that the dynamics of P2P systems still poses many exciting challenges for the future.

2 SELFISHNESS

In the first part of this paper, we presented maintenance algorithms which render a system provably robust to worst-case churn. Thereby, however, we made the implicit assumption that all peers act according to our protocols. This however might not be the case in reality: Peers are often selfish and only follow those protocols which maximize their benefit.

Selfishness is an important problem for P2P systems, since the power of P2P computing arises from the collaboration of its numerous constituent parts, the peers. In the second part of this paper, we study the effect of selfish peer behavior on P2P topologies. Concretely, we ask: How far from optimal is the performance of a system if peers act selfishly?

Concretely, we look at unstructured P2P systems—the predominant P2P architectures in today's Internet—, where a peer can select to which and to how many other peers in the network it wants to connect. With a clever choice of neighbors, a peer can attempt to optimize its lookup performance by minimizing the latencies—or more precisely, the *stretch*—to the other peers in the network. Achieving good stretches by itself is of course simple: A peer can establish links to a large number of other peers in the system. Because the memory and maintenance overhead of such a neighbor set is large, however, egoistic peers try to exploit locality as much as possible, while avoiding to store too many neighbors. It is this fundamental trade-off between the need to have small latencies and the desire to reduce maintenance overhead that governs the decisions of selfish peers.

An appropriate tool to study such selfish behavior is *game theory*. In particular, this paper studies the *Price of Anarchy* of P2P overlay creation, which is the ratio between an optimal solution obtained by perfectly collaborating participants compared to a solution generated by peers that act in an egoistic manner, optimizing their individual benefit. The importance of studying the Price of Anarchy in peer-to-peer systems stems from the fact that it quantifies the possible degradation caused by selfishness. Specifically, a low Price of Anarchy indicates that a system does not require an incentive-mechanism (such as *tit-for-tat*), because selfishness does not overly bog down the overall system performance. If the Price of Anarchy is high, however, specific cooperation incentives (whose goal is to reduce the Price of Anarchy) need to be enforced in order to ensure that the system can perform efficiently. In peer-to-peer systems therefore, the Price of Anarchy is a measure that helps explaining the necessity (or non-necessity) of cooperation mechanisms in various aspects of these systems.

The contribution of this paper is twofold. First, we show that the topologies of selfish, unstructured P2P systems can be much worse than in a scenario in which peers collaborate. More precisely, we show that the Price of Anarchy is $\Theta(\min(\alpha, n))$, where α is a parameter that captures the tradeoff between lookup performance (low stretches) and the cost of neighbor maintenance, and n is the number of peers in the system, respectively. Thereby, the upper bound $O(\min(\alpha, n))$ holds for peers located in *arbitrary* metric spaces, including the popular *growth-bounded* and *doubling metrics*. On the other hand, intriguingly, this bound is tight even in such a simple metric space as the 1-dimensional *Euclidean space*. As a second contribution, we prove that the topology of a static peer-to-peer system consisting of selfish peers may never converge to a stable state. That is, links may continuously change even in environments without *churn* (causing the network to be inherently instable).

2.1 Model

We model the peers of a P2P network as points in a *metric space* $\mathcal{M} = (V, d)$, where $d : V \times V \to [0, \infty)$ is the *distance function* which describes the underlying latencies between all pairs of peers.

The effects of selfish peer behavior is studied from a *game-theoretic* perspective. We consider a set of n peers

$$V = \{\pi_0, \pi_1, \ldots, \pi_{n-1}\}.$$

A peer can choose to which subset of other peers it wants to store pointers (IP addresses). Formally, the *strategy space* of a peer π_i is given by $S_i = 2^{V \setminus \{\pi_i\}}$, and we will refer to the actually chosen links as π_i's *strategy* $s_i \in S_i$. We say that π_i *maintains or establishes a link* to π_j if $\pi_j \in s_i$. The combination of all peers' strategies, i.e., $s = (s_0, ..., s_{n-1}) \in S_0 \times \cdots \times S_{n-1}$, yields a (directed) graph $G[s] = (V, \cup_{i=0}^{n-1}(\{\pi_i\} \times s_i))$, which describes the resulting P2P topology.

Selfish peers exploit *locality* in order to maximize their lookup performance. Concretely, a peer aims at minimizing the *stretch* to all other peers. The stretch between two peers π and π' is defined as the shortest distance between π and π' using the links of the resulting P2P topology G divided by the direct distance, i.e., for a topology G, $stretch_G(\pi, \pi') = d_G(\pi, \pi')/d(\pi, \pi')$. Clearly, it is desirable for a peer to have low stretch to other peers in order to keep its latency small. By establishing a link to all peers in the system, a peer reaches every peer with minimal stretch 1, and the potential lookup performance is optimal. However, storing and especially maintaining a large number of links is expensive.[3] Hence, the individual cost $c_i(s)$ incurred at a peer π is composed not only of the stretches to all other peers, but also of its *degree*, i.e., the number of its neighbors:

$$c_i(s) = \alpha \cdot |s_i| + \sum_{i \neq j} stretch_{G[s]}(\pi_i, \pi_j).$$

Note that this cost function captures the classic P2P trade-off between the need to minimize latencies and the desire to store and maintain only few links, as it has been addressed by many existing systems, for example Pastry [29]. Thereby, the relative importance of degree costs versus stretch costs is expressed by the parameter α.

The objective of a selfish peer is to minimize its individual cost. In order to evaluate the topologies constructed by such selfish peers—and compare them with the topologies achieved by collaborating peers—, we use the notion of a *Nash equilibrium*. A P2P topology constitutes a Nash equilibrium if no peer can reduce its individual cost by changing its set of neighbors given that the connections of all other peers remain the same. More formally, a (pure) Nash equilibrium is a combination of strategies s such that, for each peer π_i, and for all alternative strategies s' which differ only in the i^{th} component (different neighbor sets for peer π_i), $c_i(s) \leq c_i(s')$. This means that in a Nash equilibrium, no peer has an incentive to change its current set of neighbors, that is, Nash equilibria are *stable*.

While peers try to minimize their individual cost, the system designer is interested in a good overall quality of the P2P network. The *social cost* is the sum of all peers' individual costs, i.e.,

$$C(G) = \sum_i c_i = \alpha|E| + \sum_{i \neq j} stretch_G(\pi_i, \pi_j).$$

The lower this social cost, the better is the system's performance.

Determining the parameter α in real unstructured peer-to-peer networks is an interesting field for study. As mentioned, α measures the relative importance of low stretches compared to the peers' degrees, and thus depends on the system or application: For example, in systems with many lookups where good response times are crucial, α is smaller than in distributed archival storage systems consisting mainly

[3]For instance, the maintenance of a link may involve periodic pings to verify whether the neighbor is still alive.

of large files.[4] In the sequel, we denote link and stretch costs by $C_E(G) = \alpha|E|$ and $C_S(G) = \sum_{i \neq j} stretch_G(\pi_i, \pi_j)$, respectively.

Typically, a given distribution of peers in a metric space can result in different Nash equilibria, depending on the order in which peers change their links. To gain an understanding of the impact of selfishness on the social cost, we are particularly interested in the social cost of the *worst* possible Nash equilibrium. That is, we study topologies in which no selfish peer has an incentive to change its neighbors, but in which all peers together could be much better off if they collaborated. More precisely and using the terminology of game theory, we are interested in the *Price of Anarchy*, the ratio between the social cost of the worst Nash equilibrium and the social cost of the optimal topology.

2.2 Related Work

The lack of cooperation in traditional P2P file-sharing systems has been well-documented over the last years [5, 36], and research on the causes and possible counter-measures is very active, e.g., [8] and [17]. Most of the current literature focuses on the issue of free resource consumption, *freeriding*. In contrast, the impact of other aspects of selfishness has received much less attention. In fact, to the best of our knowledge, this is the first paper to take a step towards studying the consequences of selfish neighbor selection on the topologies of P2P networks.

The first paper to study the creation of networks from a game-theoretic point of view is due to Fabrikant et al. [14]. In this paper, the authors analyze the Internet's architecture as built by economic agents, e.g., by Internet providers or *autonomous systems*. Recent subsequent work on network creation in various settings includes [6, 9, 12]. In contrast to all these works, our model takes into account many of the intrinsic properties of P2P systems. For instance, it captures the important *locality properties* of P2P systems, i.e., the desire to reduce the latencies (expressed as the stretch) experienced when performing look-up operations. Furthermore, the fact that a peer can decide to which other peers it wishes to store pointers yields a scenario with *directed* links.

Building structured systems that explicitly exploit locality properties has been a flourishing research area in networking and P2P computing (e.g. [2, 29, 35]). In early literature on distributed hash tables (DHT), the major measure of system quality has been the number of hops required for look-up operations. While this hop-distance is certainly of importance, it has been argued that the delay of communication (i.e., the stretch between pairs of peers) is a more relevant quality measure. Based on results achieved in [24], systems such as [2, 4, 29, 37] guarantee a provably bounded stretch with a limited number of links per peer. All of these systems are structured and peers are supposed to participate in a carefully predefined topology. Our paper complements this line of research by analyzing topologies as they are created by *selfish peers*, which are interested only in optimizing their *individual* trade-off between locality and maintenance overhead.

2.3 Price of Anarchy

The Price of Anarchy is a measure to bound the degradation of a globally optimal solution caused by selfish individuals. In this section, we show that the topologies created by selfish peers deteriorate more (compared to collaborative networks) as the cost of maintaining links becomes more important (larger α). Concretely, in Section 2.3.1 we prove that for *arbitrary* metrics—thus, including the important and well-studied *growth-bounded* and *doubling metrics*—, the Price of Anarchy never exceeds $O(\min(\alpha, n))$. We then show in Section 2.3.2 that this bound is tight even in the "simplest" metric space, the 1-dimensional Euclidean space.

[4]If α is in the order of $\Theta(\sqrt{n})$, for instance, P2P topologies in which the latency stretch between all pairs of peers is bounded by a constant, and in which every peer has at most degree $O(\sqrt{n})$ can be shown to be asymptotically optimal. This trade-off has for example been achieved by the Tulip system proposed in [2].

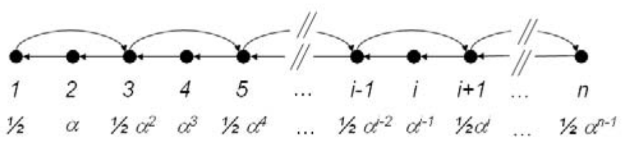

Figure 2: Example topology G where the Price of Anarchy is $\Theta(\min(\alpha, n))$ for $3.4 \le \alpha$.

2.3.1 Upper Bound

Assume the most general setting where n peers are arbitrarily located in a given metric space $\mathcal{M}$, and consider a peer π which has to find a suitable neighbor set. Clearly, the *maximal* stretch from π to any other peer π' in the system is at most $\alpha + 1$: If $stretch(\pi, \pi') > \alpha + 1$, π could establish a direct link to π', reducing the stretch from more than $\alpha + 1$ to 1, while incurring a link cost of α. Therefore, in any Nash equilibrium, no stretch exceeds $\alpha + 1$. Because there are at most $n(n-1)$ directed links (from each peer to all remaining peers), the social cost of a Nash equilibrium is $O(\alpha n^2)$. Since the optimum social costs is clearly lower bounded by $\Omega(\alpha n + n^2)$, we have the following result.

Theorem 2.1. *For any metric space $\mathcal{M}$, the Price of Anarchy is $O(\min(\alpha, n))$.*

2.3.2 Lower Bound

We now show that there are P2P networks in with a Price of Anarchy of $\Omega(\min(\alpha, n))$, which implies that the upper bound of Section 2.3.1 is asymptotically tight. Intriguingly, the Price of Anarchy can deteriorate to $\Theta(\min(\alpha, n))$ even if the underlying latency metric describes a simple 1-dimensional Euclidean space.

Consider the topology G in Figure 2 in which peers are located on a 1-dimensional Euclidean line, and the distance (latency) between two consecutive peers increases exponentially towards the right. Concretely, peer i is located at position $\alpha^{i-1}/2$ if i is odd, and at position α^{i-1} if i is even. The peers of G maintain links as follows: All peers have a link to their nearest neighbor on the left. Odd peers additionally have a link to the second nearest peer on their right. In the following, we prove that G constitutes a Nash equilibrium. Afterwards, we derive the lower bound on the Price of Anarchy by computing the social cost of this topology.

Lemma 2.2. *The topology G shown in Figure 2 forms a Nash equilibrium for $\alpha \ge 3.4$.*

Proof. In the following, a proof sketch is given only.

We distinguish between even and odd peers. For both cases, we show that no peer has an incentive to deviate from its strategy.

Case even peers: Every even peer i needs to link to at least one peer on its left, otherwise i cannot reach the peers $j < i$. A connection to peer $i-1$ is optimal, as the stretch to all peers $j < i$ becomes 1. Observe that every alternative link to the left would imply a larger stretch to at least one peer on the left without reducing the stretch to peers on the right. Furthermore, i cannot reduce the distance to any—neither left nor right—peer by adding further links to the left. Hence, it only remains to show that i cannot benefit from adding more links to the right.

By adding a link to the right, peer i shortens the distance to *all* peers on the right. However, the cost reduction per peer decreases as a geometric series, and any such link to the right would strictly increase i's costs. To show this, we consider two cases in turn: i linking to an odd peer on the right, and i linking to an even peer on the right.

Link to an odd peer: Consider the benefit of i adding a link to its odd neighbor $i+1$. For an odd peer $j > i$, we define the *benefit* $B_{i,j}$ as the stretch cost reduction caused by the addition of the link $(i, i+1)$. We have, for $i \ge 2$,

$$\begin{aligned} B_{i,j} &= stretch_{old}(i,j) - stretch_{new}(i,j) \\ &= \frac{d(i,i-1) + d(i-1,j)}{d(i,j)} - \frac{d(i,j)}{d(i,j)} \\ &= \frac{2 - 1/\alpha}{1/2\alpha^{j-i} - 1}. \end{aligned}$$

Similarly, the savings $B_{i,j}$ for an even peer $j > i$ and $i \ge 2$ amount to $B_{i,j} = (2 - 1/\alpha)/(\alpha^{j-i} - 1)$.

Hence, for all $\alpha \geq 3.4$, the total savings B_i for peer i are less than

$$\begin{aligned} B_i &= \sum_{\text{odd } j > i} B_{i,j} + \sum_{\text{even } j > i} B_{i,j} \\ &< \sum_{\delta=1}^{\infty} \frac{2-\frac{1}{\alpha}}{\frac{1}{2}\alpha^{2\delta-1}-1} + \sum_{\delta=1}^{\infty} \frac{2-\frac{1}{\alpha}}{\alpha^{2\delta}-1} \\ &< \frac{4\alpha^2-1}{\alpha^2-1} \underset{(\alpha \geq 3.4)}{<} \alpha+1. \end{aligned}$$

Therefore, the construction of link $(i, i+1)$ would be of no avail (benefit smaller than cost). Clearly, the benefit of alternative or additional links to odd neighbors on the right is even smaller.

Link to an even peer: A link to an even peer $j > i$ entails a stretch 1 to the corresponding peer instead of $stretch_{old}(i,j) = (\alpha^j - \alpha^{j-1} + \alpha^{i-1} - \alpha^{i-2})/(\alpha^{j-1} - \alpha^{i-1}) < \alpha + 1$ for $\alpha > 2$. However, the stretch from i to all other peers remains unchanged, since the path $i \rightsquigarrow (i-1) \rightsquigarrow (i+1)$ is shorter than $i \rightsquigarrow (i+2) \rightsquigarrow (i+1)$: $\alpha^{i-1} - \alpha^{i-2}/2 + \alpha^i/2 - \alpha^{i-2}/2 < \alpha^{i+1} - \alpha^{i-1} + \alpha^{i+1} - \alpha^i/2$ for $\alpha > 1$. Therefore, an even peer i has no incentive to build links to any even peer on its right.

Case odd peers: The proof that an odd peer i has no incentive to change its neighbor set is similar to the proof for even peers, and it is omitted here. □

Having verified that the topology of Figure 2 is a Nash equilibrium, its social cost can be computed.

Lemma 2.3. *The social cost $C(G)$ of the topology G shown in Figure 2 is $C(G) \in \Theta(\alpha n^2)$.*

Proof. Clearly, the link costs of topology G are $C_E(G) \in \Theta(\alpha n)$. But since the stretch from an odd peer i to an even peer $j > i$ and the stretch between two even peers i and $j > i$ are $stretch(i,j) > \alpha/2$ (for $\alpha > 2$), the stretch costs are $C_S \in \Theta(\alpha n^2)$. □

Theorem 2.4. *The Price of Anarchy of the peer topology G shown in Figure 2 is $\Theta(\min(\alpha, n))$.*

Proof. The theorem follows from Theorem 2.1 and Lemmas 2.2 and 2.3, and by the observation that the optimal social cost of a topology connecting the peers in Figure 2 is upper bounded by $O(\alpha n + n^2)$. For the latter, assume that there are no links in Figure 2. If every peer connects to the nearest peer on its left and to the nearest peer on its right, there are $2(n-1)$ links, and all stretches are 1. Thus, the social cost of this resulting topology $\widetilde{G}$ is $C(\widetilde{G}) = \alpha \cdot 2(n-1) + n(n-1) \in O(\alpha n + n^2)$. The optimal social cost is at most the social cost of $\widetilde{G}$. □

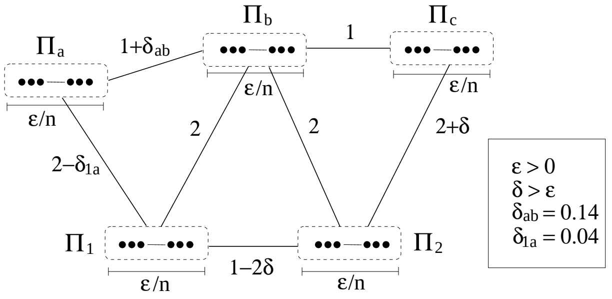

Figure 3: Instance I_k has no pure Nash equilibrium when $\alpha = 0.6k$, where $k = n/5$. The number of peers in each cluster is k.

2.4 Existence of Nash Equilibria

In this section, we show that a system of selfish peers may never converge to a stable state, even in the absence of churn, mobility, or other sources of dynamism. Interestingly, this result even holds if we assume latencies to form simple metric spaces, such as a 2-dimensional Euclidean space.

Theorem 2.5. *Regardless of the magnitude of α, there are metric spaces $\mathcal{M}$, for which there exists no pure Nash equilibrium, i.e. certain P2P networks cannot converge to a stable state. This is the case even if $\mathcal{M}$ is a 2-dimensional Euclidean space.*

Instead of presenting the formal proof, we attempt to highlight the main ideas only. Assume that the parameter α is a multiple of 0.6, i.e., $\alpha_k = 0.6k$ for an arbitrary integer $k > 0$. Given a specific k, we show that the 2-dimensional Euclidean instance I_k

of Figure 3 has no pure Nash equilibrium. Specifically, I_k constitutes a situation in which there are peers $\pi_1 \in \Pi_1$ and $\pi_2 \in \Pi_2$ that continue to deviate to a better strategy ad infinitum, i.e., the system cannot converge.

The n peers of instance I_k are grouped into five clusters Π_1, Π_2, Π_a, Π_b, and Π_c, each containing $k = n/5$ peers. Within a cluster, peers are located equidistantly on a line, and each cluster's diameter is ϵ/n, where $\epsilon > 0$ is an arbitrarily small constant. The *inter-cluster distance* $d(\Pi_i, \Pi_j)$ between Π_i and Π_j is the minimal distance between any two peers in the two clusters. Distances not explicitly defined in Figure 3 follow implicitly from the constraints imposed by the Euclidean plane. A link from a peer $\pi_i \in \Pi_i$ to a peer $\pi_j \in \Pi_j$ is denoted by ℓ_{ij}. Clusters Π_a, Π_b, and Π_c are called *top-clusters* and finally, δ denotes an arbitrarily small positive number such that $\delta > 10\epsilon$.

The proof unfolds in a series of lemmas that characterize the structure of the resulting graph $G[s]$ if the strategies s form a Nash equilibrium in I_k. First, it can be shown that in $G[s]$, two peers in the same cluster are always connected by a path that does not leave the cluster. The reason is that in the absence of such a link, there is a stretch of at least $\frac{2-2\delta}{\epsilon/n}$ between each pair of peers in the same cluster. By constructing an intra-cluster link at cost α, a peer can significantly reduce these stretches, rendering such a link worthwhile. Furthermore, it can be shown that there exists exactly one link in both directions between clusters Π_a and Π_b, Π_b and Π_c, as well as between clusters Π_1 and Π_2. In all cases, the argument is based on the fact that without such a link, the sum of the stretch between a peer in one cluster to the peers in the neighboring cluster would exceed $k(2 - 2\delta)$. Because a single link to a peer in this neighboring cluster can reduce each stretch to $1 + \epsilon$, the cost of connecting directly to a peer in the neighboring cluster is less than $\alpha + k(1 + \epsilon)$, rendering the construction of such a link worthwhile. A third structural characteristic that can be derived for any Nash equilibrium is that there is at most one directed link from a cluster Π_i to peers in a cluster Π_j. Since ϵ is small and all peers are linked within their clusters, peer $\pi_i \in Pi_i$ reduces its cost by dropping its link to cluster Π_j, if another peer in π_i's cluster has a link to a peer in Π_j.

To preserve connectivity, some peers in Π_1 and Π_2 must have links to top-peers. Based on the above observation that there is at most one link between two clusters in each direction, Lemma 2.6 further narrows down the set of possible strategies for connecting to top-peers.

Lemma 2.6. *In any Nash equilibrium of instance I_k, it holds that*

i) Neither peers in Π_1 nor Π_2 select three links to top-peers.

ii) There exists peer $\pi_1 \in \Pi_1$ that establishes a link to Π_a.

iii) There is exactly one link from cluster Π_2 to either cluster Π_b or Π_c, but no link to Π_a.

Correctness of all three properties is proven by verifying that there exists some node $\pi_1 \in \Pi_1$ or $\pi_2 \in \Pi_2$ has an incentive to change its strategy in case the property is not satisfied. If, for instance, there are two peers $\pi_2, \pi_2' \in \Pi_2$ that simultaneously maintain links to both Π_b and Π_c, (e.g. π_2 to Π_b and π_2' to Π_c, thus violating case iii)), π_2' can lower its costs if it drops its links to Π_c. Intuitively, this holds because the sum of the stretches $\sum_{\pi_c \in \Pi_c} stretch(\pi_2', \pi_c)$ entailed by the indirection $\pi_2' \rightsquigarrow \pi_2 \rightsquigarrow \Pi_b \rightsquigarrow \Pi_c$ does not justify the additional cost α when maintaining ℓ_{2c}.

It can be shown that only the six structures depicted in Figure 4 remain valid candidates for Nash topologies. In each scenario, however, at least one peer benefits from deviating from its current strategy.

Case 1: In this case, a peer $\pi_1 \in \Pi_1$ can reduce its cost by adding a link ℓ_{1b} to a peer in Π_b.

Case 2: If the only outgoing link from Π_1 to a top-cluster is to cluster Π_a, the peer $\pi_2 \in \Pi_2$ maintaining the look to Π_c can be shown to profit from switching

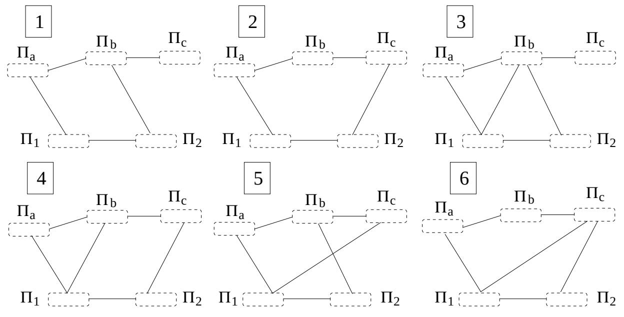

Figure 4: Candidates for a Nash equilibrium.

its link from Π_c to Π_b.
Case 3: The availability of ℓ_{1b} changes the optimal choice of $\pi_2 \in \Pi_2$. Unlike in the previous case, π_2 prefers linking to Π_c instead of Π_b.
Case 4: Due to the existence of a link from a peer $\pi_2 \in \Pi_2$ to Π_c, the peer $\pi_1 \in \Pi_1$ with the link to Π_b has an incentive to drop this link ℓ_{1b}.
Case 5: In this case, the peer $\pi_1 \in \Pi_1$ reduces its cost by replacing its link to cluster Π_c with a link to Π_b and.
Case 6: Finally, this case is similar to Case 4 in the sense that $\pi_1 \in \Pi_1$ with the link to Π_b has an incentive to remove ℓ_{1c}.

This proof highlights how the system is ultimately trapped in an infinite loop of strategy changes, without ever converging to a stable situation. There is always at least one peer which can reduce its cost by changing its strategy. For instance, the following sequence of topology changes could repeat forever (cf. Figure 4): $1 \rightsquigarrow 3 \rightsquigarrow 4 \rightsquigarrow 2 \rightsquigarrow 1 \rightsquigarrow 3 \ldots$ In other words, selfish peers will not achieve a stable network topology.

2.5 Conclusion

In the second part of this paper, we have seen that selfishness can have a large impact on P2P systems. For instance, selfishness can cause the network to be inherently instable even if there are no membership changes. This emphasizes the fact that a successful system in practice must be able to cope with both *churn and selfishness.*

Of course, many additional aspects of selfishness have to be taken into account (e.g., not contributing any upload bandwidth). While we were able to come up with a mechanism to tame dynamic peers, handling selfishness seems to be hard: How can one ensure that peers select their neighbors in a more collaborative (or equivalently: globally better) manner? How can a peer's behavior be efficiently verified? We leave this issue for future research.

References

[1] Karl Aberer. P-Grid: A Self-Organizing Access Structure for P2P Information Systems. In *Proc. 9th Int. Conference on Cooperative Information Systems (CoopIS)*, pages 179–194, 2001.

[2] I. Abraham, A. Badola, D. Bickson, Dahlia Malkhi, Sharad Maloo, and Saar Ron. Practical Locality-Awareness for Large Scale Information Sharing. In *IPTPS*, 2005.

[3] Ittai Abraham, Baruch Awerbuch, Yossi Azar, Yair Bartal, Dahlia Malkhi, and Elan Pavlov. A Generic Scheme for Building Overlay Networks in Adversarial Scenarios. In *Proc. 17th Int. Symp. on Parallel and Distributed Processing (IPDPS)*, page 40.2, 2003.

[4] Ittai Abraham, Dahlia Malkhi, and Oren Dobzinski. LAND: Stretch $(1 + e)$ Locality Aware Networks for DHTs. In *SODA*, 2004.

[5] E. Adar and B. Huberman. Free Riding on Gnutella. *First Monday*, 5(10), 2000.

[6] Susanne Albers, Stefan Eilts, Eyal Even-Dar, Yishay Mansour, and Liam Roditty. On Nash Equilibria for a Network Creation Game. In *SODA*, 2006.

[7] Keno Albrecht, Ruedi Arnold, Michael Gähwiler, and Roger Wattenhofer. Aggregating Information in Peer-to-Peer Systems for Improved Join and Leave. In *4th IEEE Int. Conference on Peer-to-Peer Computing (P2P)*, 2004.

[8] Nazareno Andrade, Miranda Mowbray, Aliandro Lima, Gustavo Wagner, and Matei Ripeanu. Influences on Cooperation in BitTorrent Communities. In *SIGCOMM*, 2005.

[9] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The Price of Stability for Network Design with Fair Cost Allocation. In *FOCS*, 2004.

[10] James Aspnes and Gauri Shah. Skip Graphs. In *Proc. 14th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 384–393, 2003.

[11] Baruch Awerbuch and Christian Scheideler. The Hyperring: A Low-Congestion Deterministic Data Structure for Distributed Environments. In *Proc. 15th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 318–327, 2004.

[12] J. Corbo and D. C. Parkes. The Price of Selfish Behavior in Bilateral Network Formation. In *PODC*, 2005.

[13] G. Cybenko. Dynamic Load Balancing for Distributed Memory Multiprocessors. *Journal on Parallel Distributed Computing*, 7:279–301, 1989.

[14] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a Network Creation Game. In *PODC*, 2003.

[15] A. Fiat and J. Saia. Censorship Resistant Peer-to-Peer Content Addressable Networks. In *Proc. 13th Symp. on Discrete Algorithms (SODA)*, 2002.

[16] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *Proc. 4th USENIX Symp. on Internet Technologies and Systems (USITS)*, 2003.

[17] Seung Jun and Mustaque Ahamad. Incentives in BitTorrent Induce Free Riding. In *SIGCOMM*, 2005.

[18] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proc. of ACM ASPLOS*, November 2000.

[19] Fabian Kuhn, Stefan Schmid, and Roger Wattenhofer. A Self-Repairing Peer-to-Peer System Resilient to Dynamic Adversarial Churn. In *Proc. 4th Itl. Workshop on Peer-to-Peer Systems (IPTPS). Springer Lecture Notes on Computer Science (LNCS), Copyright 2005.*, 2005.

[20] Xiaozhou Li, Jayadev Misra, and C. Greg Plaxton. Active and Concurrent Topology Maintenance. In *Proc. 18th Ann. Conference on Distributed Computing (DISC)*, 2004.

[21] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *Proc. 21st Ann. Symp. on Principles of Distributed Computing (PODC)*, pages 183–192, 2002.

[22] Thomas Moscibroda, Stefan Schmid, and Roger Wattenhofer. On the Topologies Formed by Selfish Peers. In *Proc. 5th Itl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.

[23] David Peleg and Edi Upfal. The Token Distribution Problem. *SIAM Journal on Computing*, 18(2):229–243, 1989.

[24] C. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *SPAA*, 1997.

[25] C. G. Plaxton. Load Balancing, Selection and Sorting on the Hypercube. In *Proc. 1st Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 64–73, 1989.

[26] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Proc. 9th Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 311–320, 1997.

[27] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content Addressable Network. In *Proc. of ACM SIGCOMM 2001*, 2001.

[28] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling Churn in a DHT. In *Proc. USENIX Ann. Technical Conference*, 2004.

[29] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. In *IFIP/ACM Int. Conf. Dist. Sys. Platforms (Middleware)*, 2001.

[30] J. Saia, A. Fiat, S. Gribble, A. Karlin, and S. Saroiu. Dynamically Fault-Tolerant Content Addressable Networks. In *Proc. 1st Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[31] Behrooz A. Shirazi, Krishna M. Kavi, and Ali R. Hurson. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Science Press, 1995.

[32] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM Conference*, 2001.

[33] R. van Renesse and A. Bozdog. Willow: DHT, Aggregation, and Publish/Subscribe in One Protocol. In *Proc. 3rd Int. Workshop on Peer-To-Peer Systems (IPTPS)*, 2004.

[34] Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Transactions on Computing Systems*, 21(2):164–206, 2003.

[35] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: A Lightweight Network Location Service without Virtual Coordinates. In *SIGCOMM*, 2005.

[36] M. Yang, Z. Zhang, X. Li, and Y. Dai. An Empirical Study of Free-Riding Behavior in the Maze P2P File Sharing System. In *IPTPS*, 2005.

[37] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE J. Selected Areas in Comm.*, 22(1), 2004.