Distrib. Comput.
DOI 10.1007/s00446-010-0099-z

Towards worst-case churn resistant peer-to-peer systems

Fabian Kuhn · Stefan Schmid · Roger Wattenhofer

Received: 24 March 2009 / Accepted: 23 February 2010
© Springer-Verlag 2010

Abstract Until now, the analysis of fault tolerance of peer-to-peer systems usually only covers random faults of some kind. Contrary to traditional algorithmic research, faults as well as joins and leaves occurring in a worst-case manner are hardly considered. In this article, we devise techniques to build dynamic peer-to-peer systems which remain fully functional in spite of an adversary who continuously adds and removes peers. We exemplify our algorithms on hypercube and pancake topologies and present a system which maintains small peer degree and network diameter.

Keywords Churn · Dynamic networks · Fault-tolerance · Overlay network · Peer-to-peer

1 Introduction

Due to the rise of peer-to-peer (p2p) systems, sensor networks, and mobile ad hoc networks, logical networks, or *overlay networks*, are becoming more and more wide spread. A major complication in these networks is that they can be highly dynamic, which requires fast and robust recovery mechanisms.

Today, the analysis of fault tolerance of p2p systems usually only covers random faults of some kind. Contrary to traditional algorithmic research, faults as well as joins and leaves occurring in a worst-case manner in p2p systems are hardly considered. Moreover, most fault tolerance analyses are static in the sense that only a functionally bounded number of random peers can be crashed. After removing a few peers the system is given sufficient time to recover again. The much more realistic dynamic case where faults steadily occur has not found much attention.

This article studies dynamic worst-case joins and leaves. We think of an *adversary* who, in contrast to related literature, cannot be fooled by any kind of randomness, that is, our guarantees not only hold probabilistically (e.g., given that the adversary does not know the current random bits) but *deterministically*. Henceforth, we will use the term "adversarial" to denote such worst-case behavior.

The adversary can choose which peers to crash and how peers join. Moreover, the adversary does not have to wait until the system is recovered before crashing the next batch of peers. Instead, the adversary can constantly crash peers while the system is trying to stay alive. Indeed, the (structured) peer-to-peer system presented in this article is never fully repaired but always fully functional. Our system is resilient against an adversary who continuously attacks the "weakest part" of the system. Such an adversary could for example insert a crawler into the p2p system, learn the topology, and then repeatedly crash selected peers, in an attempt to partition the network. Our system counters such an adversary by continuously moving the remaining or newly joining peers towards the sparse areas.

A preliminary version of this work has been presented at the 4th International Workshop on Peer-To-Peer Systems (IPTPS 2005) [28] and the 14th IEEE International Workshop on Quality of Service (IWQoS 2006) [27].

F. Kuhn
Faculty of Informatics, University of Lugano,
6904 Lugano, Switzerland
e-mail: fabian.kuhn@usi.ch

S. Schmid (✉)
Deutsche Telekom Laboratories (T-Labs),
Technische Universität Berlin, 10587 Berlin, Germany
e-mail: stefan@net.t-labs.tu-berlin.de

R. Wattenhofer
Computer Engineering an Networks Laboratory,
ETH Zurich, 8092 Zurich, Switzerland
e-mail: wattenhofer@tik.ee.ethz.ch

Of course, we cannot allow our adversary to have unbounded capabilities. In any constant time interval, the adversary can at most add and/or remove $O(\Delta)$ peers, Δ being the current peer degree in the system. Clearly, otherwise network partitions cannot be avoided. This model covers an adversary who repeatedly takes down machines by a distributed denial of service attack, however only a $\Theta(\Delta)$ many machines at each point in time.

We present techniques to design peer-to-peer systems that are provably worst-case resilient against such an adversary. Concretely, in this article, we develop two different systems achieving an optimal robustness of $\Theta(\Delta)$ using these techniques, one based on a hypercube graph and one based on a pancake graph. The basic structure of the hypercube p2p system is a *simulated 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. After a number of joins and leaves, some peers may have to change to another hypercube node such that up to constant factors, all hypercube nodes have the same cardinality 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 nodes can be seen as a *dynamic token distribution problem* on the hypercube. Each node of a graph (hypercube) has a certain number of tokens, and 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 hypercube's dimension accordingly.

As a second example, we will also show how to adopt these techniques in a pancake graph. Again, we simulate the nodes in the pancake graph by multiple peers, and devise corresponding token distribution and information aggregation algorithms. It turns out that our basic ideas also work here, although some additional reasoning regarding the topological simulation is required.

1.1 Model

This article considers a setting where a maintenance algorithm has to ensure certain topological properties despite concurrent joins and leaves orchestrated by an adversary. We assume that all peers act according to our protocol, that is, they are for example not selfish. The adversary only decides when and where (in the network) peers join and leave. We focus on the more common *fail-stop fault model* here, and leave the study of Byzantine faults for future research.

Our maintenance 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 adversary model. In our synchronous message passing model, each peer can send a message to all its neighbors in each round. In the following, we will refer to an adversary who can perform J arbitrary joins and and L arbitrary leaves (crashes) in each interval of λ rounds by $\mathcal{A}(J, L, \lambda)$.

Note that throughout this article—in contrast to other literature which often uses the terms "nodes" and "peers" interchangeably—we will refer to the vertices of a to be simulated graph as *nodes*, and to the machines simulating these nodes as *peers*.

1.2 Related work

On the one hand, p2p systems are potentially more resilient compared to alternative systems as they do not rely on central servers which constitute a single point of failure. On the other hand, in p2p networks, there are frequent joins and leaves, and hence failures are more common. When designing robust p2p applications, it is beneficial to have an idea of the dynamic nature of transient and permanent failures the system will be confronted with in order to optimize system performance and to include appropriate levels of redundancy.

Unfortunately, measuring the dynamics of existing p2p systems is not always simple [20]; moreover, in this emerging area where the space of possible applications is still not well understood, it is difficult to generalize a given measurement result to entire application classes [54]. Nevertheless, there have been several measurement studies of p2p churn. Already in 2002, Saroiu et al. [47] reported on the dynamic nature of Napster and Gnutella, and Sen et al. [50] analyzed flow-level data of a large ISP to estimate churn. The availability of the *Overnet* system has been studied in [8] by probing crawled hosts; the authors showed that previous works had underestimated host availability due to a methodological limitation. Qiao et al. [41] reported a similar level of dynamics in Gnutella and Overnet. Based on *Maze* [57] measurements—one of the largest p2p systems over the China education and research network—Tian and Dai [54] pointed out that our understanding of peer dynamics is far from adequate, and that previous crawler based measurements can only yield inaccurate results. Moreover, their data reveals that newly registered peers generally have a higher turnover rate than elder ones. This observation has been confirmed by measurements on the Kad network [52,53]: peer arrivals do not follow a Poisson distribution and session times are typically not distributed exponentially. It has also been discovered that the levels of churn vary depending on the application. For instance, Guha et al. [19] showed that Skype has a higher host availability than other p2p systems. In this article, we assume a worst-case perspective and our results do not depend on any specific churn distribution. This is rather conservative when thinking of "normal" churn, but is justified in context of DoS attacks or topological viruses. Moreover, it gives the strongest possible guarantees in theory.

The effects of churn have been reported in several papers. The price of churn manifests itself in terms of dropped messages, data inconsistency, increased user-experienced latencies or increased bandwidth use [18,29,44]. It has been shown that even in stable and managed infrastructures like *PlanetLab*, there can be a significant rate of node failure due to nodes becoming extremely slow suddenly and unpredictably [43].

Scant attention is devoted to *theoretic* questions on the dynamics of p2p networks. Protocols such as Pastry [45] and CAN [42] allow for unexpected failures, and it is shown that they remain well-structured after failures occur. However, for this analysis, an ideal initial state is assumed, and it is not shown how a system can return to the initial state after the membership changes [31]. Moreover, maintenance costs can be unnecessarily high. For example, in CAN, a background stabilization process is used which introduces a constant overhead [1].

In the last few years, Awerbuch and Scheideler have proposed several interesting algorithms to render today's peer-to-peer systems more robust. For example, in Awerbuch and Scheideler [5], searchable concurrent data structures are studied where data elements can be stored on a dynamic set of nodes, e.g., in a peer-to-peer network. Their *Hyperring* data structure has degree $O(\log n)$ and requires $O(\log^3 n)$ work for insert and delete operations; search time and congestion is bounded by $O(\log n)$ with high probability, which improves on alternative structures, e.g., the deterministic *Skipnet* by Harvey and Munro [21]. The PAGODA [9] system allows heterogeneous nodes to join and leave in polylogarithmic work, which is improved by the recent SHELL [49] system.

The lack of admission control in p2p networks has triggered researchers to investigate robust join mechanisms [15]. In Scheideler [48], considers a game between a join algorithm and an adversary. The join algorithm includes both n "good" and ϵn (for some fixed constant $\epsilon < 1$) "bad" peers in a ring structure, and seeks to ensure that there is no sequence of size $\Theta(\log n)$ of consecutive peers in which at least half of the peers are malicious. The *k-rotation* strategy is presented which wins with high probability as long as $\epsilon < 1/3$. The scalable solution to join-leave attacks presented in Awerbuch and Scheideler [7] requires a robust random number generator described in Awerbuch and Scheideler, [6]. Finally, Scheideler et al. [8] have studied robust data replication strategies for distributed (but completely connected) information systems that are under a past-insider DoS attack, and proposed the so-called *Chameleon* network.

A promising class of novel p2p architectures is due to Naor and Wieder [35]. The authors describe a dynamic decomposition of a 1-dimensional continuous ID space into cells corresponding to processors, and show how to approximate de Bruijn graphs or hypercubes. In particular, their *distance halving* DHT yields an optimal tradeoff in the sense that peer degree d implies a network diameter of $O(\log_d n)$. Besides achieving a low congestion of $O((\log_d n)/n)$, their system is also provably robust against random faults. Finally, a construction for dynamic *expanders* is proposed which is based on the tessellation of the plane. This continuous-discrete approach has also been used in SHELL [49], an oblivious distributed heap network that allows for rapid joins and leaves, and which in addition to tolerating a high degree of dynamics, features robustness guarantees against certain attacks (e.g., Sybil attacks). The randomized nature of SHELL may fail however in the presence of the deterministic adversary considered in this work.

Liben-Nowell et al. [31] have analyzed the evolution of p2p systems in the face of concurrent joins and unexpected departures. They give a lower bound for the rate at which peers in the Chord system must participate to maintain the system's distributed state. For instance, they prove that if churn can be described by a Poisson distribution, a peer which receives fewer than k notifications per *half-life* will be disconnected from the network with probability at least $(1 - 1/(e-1))^k$. The half-life time period is defined as the time which elapses in a network of n live peers before n additional peers arrive, or before half of the peers depart. Their result implies that a successor list of length $\Theta(\log n)$ per peer is sufficient to ensure that a graph stays connected with high probability, as long as $\Omega(\log n)$ rounds pass before $n/2$ peers fail. It is also shown in Liben-Nowell et al. [31] that a modified version of Chord is within a logarithmic factor of the optimal rate. The authors assume that the half-life is known, and the question of how to learn the correct maintenance rate of the behavior of neighbors is left for future research. In Pandurangan et al. [36], a model similar to the one proposed in Liben-Nowell et al. [31] is considered; however, in this paper, it is assumed that a central server can direct joins to specific locations in the network, and can update old peers' neighbors when a peer leaves. In this scenario, they are able to maintain connectivity with a constant amount of space per peer. In contrast to these works, we assume a sequence of worst-case membership changes rather than a Poisson distribution. Also, we focus on feasibility of maintaining certain properties in the network; although we believe that the message complexity of our construction is good in the presence of our adversary, corresponding bandwidth lower bounds have not been derived yet and are an interesting subject for future research.

A paper by Godfrey et al. [18] has studied—using real-world traces—how to manage churn through the judicious selection of nodes are likely to remain up for a long time. By comparing different algorithms, the authors found that a uniform-random replacement strategy performs quite well. In [33], an analytical framework based on percolation theory is proposed to assess the robustness of p2p systems. Besides a probabilistic churn model, they also consider a targeted attack where nodes having high degrees are progressively removed. In the harsh model considered in our article, such a strategy is not beneficial as the adversary can always remove those peers that are believed to be "likely to remain up for a long time".

Resilience to *worst-case failures* (for the more challenging Byzantine fault model) has been studied by Fiat, Saia et al. [14,46]. The authors introduce a system where a $(1-\varepsilon)$-fraction of peers and data survives the adversarial removal of up to half of all nodes with high probability. However, in contrast to our work, the failure model is static. Moreover, the whole structure is designed with a rough a-priori knowledge of the total number of participants, and has to be rebuilt from scratch if the number of peers changes by a constant factor. Abraham et al. [1] address scalability and resilience to worst-case joins and leaves, and propose a generic overlay emulation approach for graph families such as *hypercubes*, *butterflies*, or *de Bruijn* networks. They focus on maintaining a balanced network rather than on fault-tolerance in the presence of concurrent faults. In contrast to our system, whenever a join or leave takes place, the network is given some time to adapt. To the best of our knowledge, the first paper treating concurrent worst-case joins and leaves is by Li et al. [30]. 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 where leaving peers execute an "exit" protocol which does not allow for sudden crashes.

There is also work on robust *supervised* overlay networks [25], where the p2p topology is managed by one or more special machines. In these systems, the goal is to minimize information and computational overhead at the central servers while ensuring good network degree, diameter, and expansion. While such an approach can be interesting in centralized environments such as grid computing systems or games, this solution does not scale and the network entry point constitutes a single point of failure.

Finally, our work is also related to the field of topological self-stabilization where topologies are guaranteed to recover from arbitrary states. Topological self-stabilization is a relatively young field, and researchers have only started to examine the most simple networks such as *line or ring graphs* (e.g., [10,16]). Recently, a 2-dimensional graph has been made self-stabilizing, namely the Delaunay graph (using cubic distributed runtime) [23], and there is now even a polylogarithmic time construction for *skip graphs* [22]. In some sense, these results are complementary to our work: there is no bound on the adversarial power and hence the system can become arbitrarily bad at some times; however, it is always possible to repair a connected component. Unfortunately, the constructions (e.g., [22]) often do not scale as they yield very high peer degrees (up to linear). On the other hand, making our construction self-stabilizing remains an open challenge that needs to be tackled in future research.

1.3 Our contributions

This article introduces techniques to design and provably maintain robust peer-to-peer systems resilient to ongoing and dynamic membership changes that are orchestrated by an adversary. In particular, we use our approach to construct a system based on a hypercube that tolerates $\Theta(\log n)$ worst-case joins and/or crashes per constant time interval, where n is the total number of peers currently in the system. As in other p2p systems, peers have $O(\log n)$ neighbors, implying that the achieved fault-tolerance is asymptotically optimal. The usual operations (e.g. search) take time $O(\log n)$. As another, more challenging example, we consider the pancake graph with degree and diameter $O(\log n/\log\log n)$ which tolerates $O(\log n/\log\log n)$ worst-case joins and/or crashes per constant time interval (i.e., is resilient against $\mathcal{A}(O(\log n/\log\log n), O(\log n/\log\log n), O(1))$); this fault-tolerance is again asymptotically optimal. To the best of our knowledge, these are the first two networks with optimal robustness to churn in the worst-case. Besides these two systems, we believe that our token distribution and information aggregation techniques are of independent interest.

1.4 Paper organization

The remainder of this article is organized as follows. We first describe the peer-to-peer system based on a hypercube. Section 2 introduces the algorithmic components of our construction. In Sect. 3 we use these components to build the system which is analyzed subsequently. We then extend our investigations to the pancake graph in Sect. 4. Finally, we conclude in Sect. 5.

2 Algorithmic components

In our system, 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 (i.e., its peers) whose identifier matches the first d bits of the hash-value of ID. Our maintenance algorithm for the simulated hypercube system ensures 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 a dynamic peer (token) distribution algorithm on the hypercube and an information aggregation scheme which allows the nodes to change the dimension of the hypercube. These two components—the token distribution algorithm and the information aggregation—are described in turn in this section.

Concretely, we have the following plan: We first introduce an algorithm that distributes peers evenly among the hypercube nodes and derive a bound of $\phi \leq 2J + 2L + d$ (where d is the hypercube's dimension) on the maximal discrepancy ϕ of the number of peers per node at any time against an adversary $\mathcal{A}(J, L, 1)$. Subsequently, we describe how peers can estimate the current system size well enough such that the hypercube dimension can be adapted quickly if the adversary removes or adds a large number of peers. Both these results together imply that there are never too few peers per hypercube node (threat of data loss) but also not too many (threat of scalability loss and high degree). In combination, we will derive the following main theorem:

Theorem 1 *Given an adversary who inserts and removes at most* $\mathrm{O}(\log n)$ *peers per round, there is an algorithm which ensures that* (1) *every node always has at least one peer and hence no data is lost;* (2) *each node has degree* $\Theta(\log n)$*;* (3) *the network diameter is logarithmic as well; and* (4) *unnecessary copying of data is avoided.*

2.1 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 [39] and subsequently studied in various contexts (for a general randomized scheme, see e.g., the work by Rabani et al. [42], and the recent work by Friedrich and Sauerwald [17]). 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 Shirazi et al. [51].

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 can arrive and depart *during* the execution of the token distribution algorithm. In our case, peers may join and leave the hypercube at arbitrary times, so the emphasis lies on the dynamic token distribution problem on a d-dimensional hypercube topology.

We study two types of token distribution problems: 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.

Our token distribution algorithm is based on the *dimension exchange method* [11,40]. Basically, the algorithm "cycles" continuously over the d dimensions of the hypercube, distributing the tokens in one dimension i after the other: In step s, where $i = s \bmod d$, every node $u = \beta_0 \ldots \beta_i \ldots \beta_{d-1}$ having a tokens balances its tokens with its adjacent node in dimension i, $v = \beta_0 \cdots \overline{\beta_i} \cdots \beta_{d-1}$, 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 Cybenko [11] that the described algorithm yields a perfect discrepancy $\phi = 0$ after d steps for the static fractional token distribution. In Plaxton [40], it has been shown that in the worst case, $\phi = d$ after d steps in the static integer token distribution.

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 can be "superposed", i.e., added up in the analysis at the nodes. Therefore, the following lemma holds.

Lemma 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 currently in the system.*

Proof Assume that a total amount of T tokens are distributed in two different but otherwise arbitrary ways on the d-dimensional hypercube. According to Cybenko [11], in the absence of the adversary, each node has exactly $T/2^d$ tokens after d steps. On the other hand, the token insertions and removals of the adversary that happen in-between can be treated as an independent superposition, as the corresponding operations are all linear. Thus, the number of tokens at a node only depends on the insertions and deletions of the last d rounds. □

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

Lemma 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 the fractional token distribution algorithm at time t, respectively. We have $\forall t : |\tau_v(t) - \tau_{v,f}(t)| \le d/2$.*

Proof For $t = 0$, we have $\tau_v(t) = \tau_{v,f}(t)$. For symmetry reasons, it is sufficient to show the upper bound $\tau_v(t) \le \tau_{v,f}(t) + d/2$. We first prove by induction that $\tau_v(t) \le \tau_{v,f}(t) + t/2$ at time t.

For the induction step, we consider two neighbors u and v which exchange tokens. We have

$$\begin{aligned}\tau_v(t+1) &\le \left\lceil \frac{\tau_v(t) + \tau_u(t)}{2} \right\rceil \\ &\le \left\lceil \frac{\lfloor \tau_{v,f}(t) + \frac{t}{2} \rfloor + \lfloor \tau_{u,f}(t) + \frac{t}{2} \rfloor}{2} \right\rceil \\ &\le \frac{\lfloor \tau_{v,f}(t) + \frac{t}{2} \rfloor + \lfloor \tau_{u,f}(t) + \frac{t}{2} \rfloor}{2} + \frac{1}{2} \\ &\le \tau_{v,f}(t+1) + \frac{t+1}{2}.\end{aligned}$$

The second inequality follows from the induction hypothesis and the fact that $\tau_v(t)$ and $\tau_u(t)$ are integers. Note that adding or removing tokens has no influence on the difference between τ_v and $\tau_{v,f}$ because it modifies τ_v and $\tau_{v,f}$ in the same way.

So far, we have seen that the number of integer tokens can deviate from the number of fractional tokens by at most $d/2$ after the first d steps. In order to show that this holds for all times t, we consider a fractional token distribution problem $\hat{\tau}_{v,f}$ for which $\hat{\tau}_{v,f}(t-d) = \tau_v(t-d)$. Using the above argument, we have $\tau_v(t) \le \hat{\tau}_{v,f}(t) + d/2$ and by Lemma 1, we get $\hat{\tau}_{v,f}(t) = \tau_{v,f}(t)$. This concludes the proof. □

Lemma 3 *In the presence of an adversary $\mathcal{A}(J, L, 1)$, for the integer discrepancy, it always holds that $\phi \le 2J + 2L + d$.*

Proof We show that the *fractional* discrepancy ϕ_f is bounded by $2J + 2L$. Since Lemma 2 implies that for the integer discrepancy ϕ_i it holds that $\phi_i - \phi_f \le d$ (the fractional value at a node can be $d/2$ larger or $d/2$ smaller than the integer value), the claim follows. Let $J_t \le J$ and $L_t \le L$ be the insertions and deletions that happen at the beginning of step t. First, we consider the case of joins only, i.e., $L_t = 0$. Assume that all J_t tokens are inserted at node $v = \beta_0 \ldots \beta_i \ldots \beta_{d-1}$ where $i = t \bmod d$. In what follows, all indices are implicitly modulo d. In step t, according to the token distribution algorithm, v keeps $J_t/2$ tokens and sends $J_t/2$ to node $u = \beta_0 \ldots \overline{\beta_i} \ldots \beta_{d-1}$. In the following step $t+1$, $J_t/4$ tokens are sent to nodes $\beta_0 \ldots \beta_i \overline{\beta_{i+1}} \ldots \beta_{d-1}$ and $\beta_0 \ldots \overline{\beta_i \beta_{i+1}} \ldots \beta_{d-1}$, and so on. Thus, after step $t + d - 1$, every node in the d-dimensional hypercube has the same share of $J_t/2^d$ tokens from that insertion. We conclude that a node can have at most all insertions of this step, half of the insertions of the last step, a quarter of all insertions two steps ago, and so on:

$$\underbrace{J_t + \frac{J_{t-1}}{2} + \frac{J_{t-2}}{4} + \cdots + \frac{J_{t-(d-1)}}{2^{d-1}}}_{< 2J} + \underbrace{\frac{J_{t-d}}{2^d} + \frac{J_{t-(d+1)}}{2^d} + \frac{J_{t-(d+2)}}{2^d} + \cdots}_{\text{shared by all nodes}}$$

Since $J_{t-i} \le J$ for $i = 0, 1, 2, \ldots$, we have $\phi_f \le 2J$. For the case of only token deletions, the same argument can be applied, yielding a discrepancy of at most $2L$. Finally, if there are both insertions and deletions which do not cancel out each other, we have $\phi_f \le 2J + 2L$. □

2.2 Remark on random token distribution

Next, we show that if the decision to which node to assign $\lceil \frac{a+b}{2} \rceil$ tokens and to which node to assign $\lfloor \frac{a+b}{2} \rfloor$ tokens is made uniformly at random with probability 1/2, the final (static) integer discrepancy of our algorithm is even constant in expectation.

Theorem 2 *Let X be the random variable for the final discrepancy in a d-dimensional hypercube. It holds that $\mathbb{E}[X] < 3$.*

Proof In our randomized rounding scheme, the rounding direction of each edge it determined by a perfect coin flip. Let X_i be the random variable denoting the number of incoming edges of a $(d-1-i)$-dimensional sub-cube. Since there are 2^{d-1-i} edges connecting two $(d-1-i)$-dimensional sub-cubes H_0 and H_1, X_i is *binomially* distributed: $X_i \sim Bin(2^{d-1-i}, 1/2)$. If rounding happens on every edge, the sub-cubes H_0 and H_1 differ by δ_i tokens after balancing, where $\delta_i = 2 \cdot |X_i - \mathbb{E}[X_i]|$, and the random variables δ_i are mutually independent.

Assume that in the final distribution, the maximum node v has a tokens. We show that the *average* number of tokens in the system is at least $a - \sum_{i=0}^{d-1} 2^i \mathbb{E}[\delta_i]/2^d$, by counting the average number of tokens in the biggest i-dimensional sub-cubes which contain v for $i \in [0, d]$. Obviously, the 0-dimensional sub-cube consists only of v and has a tokens in total. In the next step, this sub-cube is combined with another 0-dimensional sub-cube having $a - \delta_{d-1}$ tokens. The resulting 2-dimensional hypercube having $2a - \delta_{d-1}$ tokens is combined with a hypercube having δ_{d-2} tokens less, hence there are $4a - 2\delta_{d-1} - \delta_{d-2}$ tokens in total, and so forth. After d steps, we have $a2^d - \sum_{i=0}^{d-1} 2^i \delta_i$ tokens in the whole d-dimensional hypercube, $a - \sum_{i=0}^{d-1} 2^i \delta_i / 2^d$ on average.

It remains to calculate $\mathbb{E}[\delta_i]$, which is twice the *mean deviation* of the binomial distributed random variable X_i. In the following, we will first give a proof using a *Chernoff* argument, and then derive a better bound by a *Stirling* approximation.

We will first derive this upper bound on the mean of δ_i:

$$\mathbb{E}[\delta_i] = 2 \cdot \frac{1}{2^{2^{d-1-i}}} \cdot \sum_{j=0}^{2^{d-1-i}} \binom{2^{d-1-i}}{j} \cdot \left| j - \frac{2^{d-1-i}}{2} \right|$$
$$\overset{(1)}{\leq} 2\sqrt{\pi 2^{d-1-i}}$$

Inequality (1) is a Chernoff approximation. In fact, we need the following two facts, Fact 3 and Fact 4, as well as Lemma 4.

Fact 3 (Chernoff Lower Tail) Let $X_1, \ldots, X_N$ be independent Bernoulli variables with $\mathbb{P}[X_i = 1] = p_i$. Let $X = \sum_i X_i$ denote the sum of the X_i and let $\mu = \mathbb{E}[X] = \sum_i p_i$ be the expected value for X. For $\epsilon \in (0, 1]$,

$$\mathbb{P}[X < (1-\epsilon)\mu] < \left(\frac{e^{-\epsilon}}{(1-\epsilon)^{(1-\epsilon)}} \right)^{\mu} < e^{-\mu\epsilon^2/2}.$$

Fact 4

$$\int_0^{\infty} e^{-x^2} dx = \frac{\sqrt{\pi}}{2}.$$

Lemma 4 *Let $X \sim Bin(n, 1/2)$ be binomially distributed with parameters n and $p = 1/2$. The expectation of the deviation from the mean $n/2$ is upper bounded by*

$$\mathbb{E}[|X - n/2|] \leq \sqrt{\pi n}.$$

Proof Let p_δ denote the probability that the deviation from the mean is at least δ, that is, $p_\delta = \mathbb{P}[|X - n/2|] \geq \delta$. By symmetry, we have $p_\delta = 2 \cdot \mathbb{P}[X \leq n/2 - \delta]$. For the expected deviation of the mean, we have

$$\mathbb{E}[|X - n/2|] = \sum_{\delta=1}^{n/2} \delta \cdot \mathbb{P}[|X - n/2| = \delta] = \sum_{\delta=1}^{n/2} p_\delta. \qquad (1)$$

We can bound p_δ using Chernoff:

$$p_\delta = 2 \cdot \mathbb{P}[X \leq n/2 - \delta] \leq 2e^{-\delta^2/n}. \qquad (2)$$

Combining (1) and (2), we can bound the mean deviation by

$$\begin{aligned}\mathbb{E}[|X - n/2|] &= \sum_{\delta=1}^{n/2} p_\delta \leq 2 \cdot \sum_{\delta=1}^{n/2} e^{-\delta^2/n} \\ &< 2 \cdot \sum_{\delta=1}^{\infty} e^{-\delta^2/n} \\ &< 2 \cdot \int_{\delta=1}^{\infty} e^{-\delta^2/n} d\delta \\ &= \sqrt{\pi n}.\end{aligned}$$

The integral after the last inequality can be calculated using Fact 4 and the substitution $t = \delta\sqrt{n}$ and $d\delta = \sqrt{n}dt$. This concludes the proof.

Remark There are two minor details which are neglected for readability of the above proof. First, although the Chernoff inequality gives an upper bound only for $\mathbb{P}[X < (1-\epsilon)\mu]$, we use it for $\mathbb{P}[X \leq (1-\epsilon)\mu]$. Second, the first equation in the proof holds for even n. For odd n, the deviation from the mean is not integral. Both issues can easily be solved. $\square$

Given the bound on $\mathbb{E}[\delta_i]$, we can compute the token sum

$$\begin{aligned}\sum_{i=0}^{d-1} 2^i \mathbb{E}[\delta_i] &= 2\sqrt{\pi} \sum_{i=0}^{d-1} 2^i 2^{\frac{d-1-i}{2}} \\ &= 2\sqrt{\pi} 2^{\frac{d-1}{2}} \sum_{i=0}^{d-1} 2^{\frac{i}{2}} \\ &= 2\sqrt{\pi} 2^{\frac{d-1}{2}} \sum_{i=0}^{d-1} \left(\sqrt{2}\right)^i \\ &= 2\sqrt{\pi} 2^{\frac{d-1}{2}} \frac{\left(\sqrt{2}\right)^d - 1}{\sqrt{2} - 1} \\ &\leq \frac{\sqrt{\pi}}{\sqrt{2} - 1} 2^d.\end{aligned}$$

Thus, having a node with a tokens, the average number of tokens is at least $a - \frac{\sqrt{\pi}}{\sqrt{2}-1}$, and by symmetry, the expected final discrepancy is twice as much.

We obtain an even better bound by using *Stirling's approximation* rather than applying Chernoff. In the following, we show that we overestimated by a factor of at least π, and therefore the total discrepancy is $2 \cdot \frac{1}{\pi} \cdot \frac{\sqrt{\pi}}{\sqrt{2}-1} \doteq 2.73$.

The mean deviation MD of the symmetrical binomial distribution is given by:

$$\mathrm{MD} = 2^{-n} \sum_{k=0}^{n} \binom{n}{k} \left| k - \frac{n}{2} \right| = \begin{cases} \frac{n!!}{2(n-1)!!}, & \text{if } n \text{ odd} \\ \frac{(n-1)!!}{2(n-2)!!}, & \text{if } n \text{ even} \end{cases}$$

where $n!!$ is a double factorial, i.e.

$$n!! \equiv \begin{cases} n \cdot (n-2) \cdots 3 \cdot 1 = \frac{(n+1)!}{2^{\frac{n+1}{2}} (\frac{n+1}{2})!} & (n > 0,\ \text{odd}) \\ n \cdot (n-2) \cdots 4 \cdot 2 = 2^{\frac{n}{2}} (\frac{n}{2})! & (n > 0,\ \text{even}) \\ 1 & (n \in \{-1, 0\}) \end{cases}$$

According to (an extended version of) Stirling's approximation we have

$$\sqrt{2\pi} n^{n+1/2} e^{-n+1/(12n+1)} < n! < \sqrt{2\pi} n^{n+1/2} e^{-n+1/(12n)}.$$

Applying this leads to the following inequality which can easily be verified:

$$\mathrm{MD} = 2^{-n} \sum_{k=0}^{n} \binom{n}{k} \left| k - \frac{n}{2} \right| \le \sqrt{\frac{n}{\pi}}\ .$$

□

Note that these results do not influence our asymptotic bounds. Hence, in the following, we will not consider random rounding.

2.3 Information aggregation

Our second component is an information aggregation algorithm. Distributed aggregation is an interesting field of research, and there exist many papers on the subject [24, 26,34,37] (cf also the excellent introductory book by Peleg [38]). In the following, we concentrate on aggregation on hypercubes.

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 [3,55,56], 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 $v = \beta_0 \ldots \beta_{d-1}$ maintains an array $\Gamma_v[0 \ldots d]$, 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 $\beta_0 \ldots \beta_{d-1-i}$. Further, assume that at the beginning of each step, an adversary inserts and removes an arbitrary number of tokens at arbitrary nodes. Each node $v = \beta_0 \ldots \beta_{d-1-i} \ldots \beta_{d-1}$ then calculates the new array $\Gamma'_v[0 \ldots d]$. For this, v sends $\Gamma_v[i]$ to its adjacent node $u = \beta_0 \ldots \overline{\beta_{d-1-i}} \ldots \beta_{d-1}$, 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 $\beta_0 \ldots \beta_{d-1}$. For $i \in [1, d]$, the new estimated number of tokens in the prefix domain $\beta_0 \ldots \beta_{d-1-(i+1)}$ is given by the total number of tokens in the domain $\beta_0 \ldots \beta_{d-1-i}$ plus the total number of tokens in domain $\beta_0 \ldots \overline{\beta_{d-1-i}}$ provided by node u, that is, $\Gamma'_v[i+1] = \Gamma_v[i] + \Gamma_u[i]$.

Lemma 5 *Consider two arbitrary nodes v_1 and v_2 of the d-dimensional hypercube. The 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 $t-d$.*

Proof We prove by induction that at time $t+k$, all nodes v sharing the prefix $\beta_0 \ldots \beta_{d-1-k}$ for $k \in [0, d]$ store the same value $\Gamma_v[k]$ which represents the correct state of that sub-domain in step t.

$k = 0$: There is only one node having the prefix $\beta_0 \ldots \beta_{d-1}$, so the claim trivially holds.

$k \to k+1$: By the induction hypothesis, nodes v with prefix $\beta_0 \ldots \beta_{d-1-(k+1)}\ \beta_{d-1-k}$ share the same value $\Gamma_v[k]$ corresponding to the state of the system k steps earlier; the same holds for all nodes u with prefix $\beta_0 \ldots \beta_{d-1-(k+1)}\overline{\beta_{d-1-k}}$. In step $k+1$, all these nodes having the same prefix $\beta_0 \ldots \beta_{d-1-(k+1)}$ obviously store the value $\Gamma'_v[k+1] = \Gamma'_u[k+1] = \Gamma_v[k] + \Gamma_u[k]$. □

3 The dynamic hypercube system

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 $\mathcal{A}(d+1, d+1, 6)$ who 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, and hence no data item will ever be lost.

3.1 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

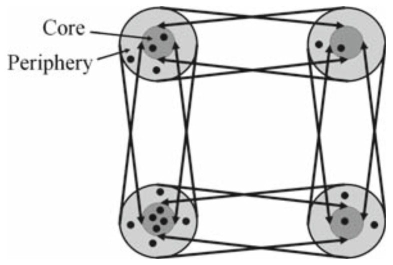

Fig. 1 A simulated 2-dimensional hypercube with four nodes, each consisting of a core and a periphery. All peers within the same node are completely connected to each other, and additionally, all peers of a node are connected to all core peers of the neighboring nodes. Only the core peers store data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

is connected to all *core* peers of the neighboring nodes (*interconnections*). 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 node in most cases.

3.2 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 time, and hence no data is lost; (2) each node 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 the 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 Sect. 2.1; 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 Sect. 2.3 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*. (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 5), the number of peers at an individual node can lie outside the threshold interval.)

While peers 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 at the beginning of the phase.

Round 1

Outline: Each node v makes a snapshot of the currently active peers, denoted by the ID set $\mathcal{S}_v$. The later rounds will only be based on these sets.

Sent messages: Each peer of a node 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

Outline: Based on the snapshot of Round 1, the core peers of a node v know the total number of peers in the node, $size(v) = |\mathcal{S}_v|$. This information is needed for the peer distribution algorithm and for the estimation of the total number of peers in the system.

Local computations: The core peers compute $size(v) = |\mathcal{S}_v|$.

Sent messages: Each peer informs its joiners about $\mathcal{S}_v$. The core peers $\mathcal{C}_v$ additionally send the number $size(v)$ to their neighboring core $\mathcal{C}_u$, where node u is v's neighbor in dimension i—the node with which v has to balance its peers in this phase. The core also exchanges the new estimated total number of peers in its prefix domains with the corresponding adjacent cores (depending on the prefix).

Round 3

Outline: At the beginning of this round, every peer within a node v knows $\mathcal{S}_v$, and the transfer for the peer distribution algorithm can be prepared. Let v again be an arbitrary node and u its adjacent node in dimension i. We assume that $size(v) > size(u)$; the case where $size(v) \leq size(u)$ is analogous and not described further here. The ID set $\mathcal{T}$ of peers that have to move from node v to node u are the $(size(v) - size(u))/2$ (arbitrarily rounded) peers in the periphery $\mathcal{P}_v$ having the smallest identifiers.
Local computations: The peers in each node v compute the new periphery $\mathcal{P}_v = \mathcal{S}_v \backslash \mathcal{C}_v$. The core remains the same.

Sent messages: All cores forward the information about the new estimated total number of peers in the system to their peripheral peers. Moreover, the core of the larger node $\mathcal{C}_v$ sends the identifiers of the to be transferred peers $\mathcal{T}$ to $\mathcal{C}_u$, and the number $(size(v) - size(u))/2$ to the new periphery $\mathcal{P}_v$.

Round 4

Outline: The transfer for the peer distribution algorithm is continued. Moreover, this round prepares the dimension reduction if necessary.

Sent messages: The core $\mathcal{C}_u$ informs the peers in $\mathcal{T}$ about all neighboring cores $\mathcal{C}_{u_j}$, where u_j is the neighbor of u in dimension j for $j \in [0, d-1]$, about $\mathcal{C}_u$ itself, about $\mathcal{S}_u$ and about its peripheral peers $\mathcal{P}_u$. Additionally, $\mathcal{C}_u$ informs its own periphery $\mathcal{P}_u$ about the newcomers $\mathcal{T}$.

If the estimated total number of peers in the system is beyond the threshold, the core peers of a node $\overline{v}$ 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 $\underline{v}$.

Round 5

Outline: 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.

Local computations: Given the number $(size(v) - size(u))/2$, the peripheral peers $\mathcal{P}_v$ can compute the set $\mathcal{T}$ selecting the $(size(v) - size(u))/2$ smallest elements in $\mathcal{P}_v$. From this, the new periphery $\mathcal{P}_v = \mathcal{P}_v \backslash \mathcal{T}$ is computed. Analogously, the peers in node u (including $\mathcal{T}$) can compute the new periphery $\mathcal{P}_u = \mathcal{P}_u \cup \mathcal{T}$.

Then, all peers of each node v calculate the new core $\mathcal{C}_v^{new}$: it consists of the peers of the old core which have still been alive in Round 1, i.e., $\mathcal{C}_v^{old} = \mathcal{C}_v \cap \mathcal{S}_v$, plus the $2d+3-|\mathcal{C}_v \cap \mathcal{S}_v|$ smallest IDs in the new periphery $\mathcal{P}_v$, denoted by $\mathcal{C}_v^{\triangle}$. Hence, the new core is given by $\mathcal{C}_v^{new} = \mathcal{C}_v^{old} \cup \mathcal{C}_v^{\triangle}$, and the new periphery by $\mathcal{P}_v^{new} = \mathcal{P}_v \backslash \mathcal{C}_v^{\triangle}$.

If the hypercube has to grow in this phase, the smallest $2d+3$ peers in the new periphery $\mathcal{P}_{\underline{v}}^{new}$ become the new core of the expanded node, $\mathcal{C}_{\overline{v}}$. Half of the remaining peripheral peers, the ones with the smaller identifiers, build the new periphery $\mathcal{P}_{\overline{v}}$, and the other half becomes $\mathcal{P}_{\underline{v}}$. All these operations can be computed locally by every peer.

Sent Messages: The old core $\mathcal{C}_v^{old}$ informs all its neighboring nodes (i.e., their old cores) about the new core $\mathcal{C}_v^{new}$. Moreover, $\mathcal{C}_v^{old}$ sends its data items to the peers in $\mathcal{C}_v^{\triangle}$.

If the hypercube is about to grow, $\mathcal{C}_{\underline{v}}^{old}$ sends the necessary data items to the core peers of the new node, $\mathcal{C}_{\overline{v}}$. Moreover, $\mathcal{C}_{\underline{v}}^{old}$ informs its neighboring (old) cores about the IDs of its expanded core $\mathcal{C}_{\overline{v}}$.

If the hypercube is about to shrink, all cores $\mathcal{C}_{\underline{v}}^{old}$ inform their periphery about the peers arriving from the expanded node and the peers in the expanded node about the new core $\mathcal{C}_{\underline{v}}^{new}$ and its periphery. $\mathcal{C}_{\underline{v}}^{old}$ also copies the data items of $\mathcal{C}_{\overline{v}}^{old}$ to the peers $\mathcal{C}_{\underline{v}}^{\triangle}$.

Round 6

Outline: The new cores are built and the dimension change is accomplished if necessary.

Local computations: If the hypercube has been reduced, every peer can now compute the new periphery $\mathcal{P}_{\underline{v}}$.

Sent messages: The old core $\mathcal{C}_v^{old}$ forwards the information about the new neighboring cores to the peers $\mathcal{C}_v^{\triangle} \cup \mathcal{P}_v$.

If the hypercube has grown, $\mathcal{C}_{\underline{v}}^{old}$ forwards the expanded cores of its neighboring nodes to *all* peers in its expanded node $\overline{v}$. Note that this requires that $\mathcal{C}_{\underline{v}}^{old}$ remembers the peripheral peers that have been transferred to $\overline{v}$ in Round 5.

Theorem 5 *Given an adversary $\mathcal{A}(d+1, d+1, 6)$ who 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.*

Proof We first consider a simpler system without the separation into core and periphery, where the maintenance algorithm simply runs the peer distribution algorithm and the information aggregation algorithm to count the total number of peers in the system, and expands or reduces the hypercube with respect to the thresholds $LT = 8d + 16$ and $UT = 40d + 80$. Moreover, we assume that these operations are performed in quiet phases, where the adversary removes at most $d+1$ and adds at most $d+1$ peers only in-between.

For this simpler system, it holds that every node in the simulated d-dimensional hypercube has at least $3d+10$ and at most $45d+86$ peers at every moment of time. Moreover, after the hypercube has changed its dimension from d_{old} to d_{new}, the dimension will remain stable for at least $2d_{new}+1$ phases.

The cases where the average number of peers per node μ falls beyond the lower threshold $8d_{old}+16$ or exceeds the upper threshold $40d_{old}+80$ are studied in turn. According to Lemma 5, such an event will lead to a dimension change with a delay of d_{old} phases only. We prove that after the change, for the average number of peers per node it holds $\mu \in [8d_{new}+16, 40d_{new}+80]$ for at least $d_{new}+1$ phases. The dimension remains stable for at least $2d_{new}+1$ phases which implies—together with Lemma 3—that the discrepancy before the next change is limited by $2(d_{new}+1) + 2(d_{new}+1) + d_{new} = 5d_{new}+4$.

Case $\mu < 8d + 16$: At time $t - d_{old}$, it held that $\mu < 8d_{old}+16$ while at time $t - d_{old} - 1$ we had $\mu \geq 8d_{old}+16$. In $d_{old}+1$ phases, there are at most $(d_{old}+1)(d_{old}+1) = d_{old}^2 + 2d_{old} + 1$ leaves, so $\mu \geq 8d_{old} + 16 - \frac{d_{old}^2+2d_{old}+1}{2^{d_{old}}} > 8d_{old}+14$ before merging. Clearly, there must be a node with more than $8d_{old} + 14$ peers, hence, given the discrepancy of $5d_{old} + 4$ (cf Lemma 3), every node has more than $3d_{old} + 10$ peers before merging.

What about the maximum? At time $t - d_{old}$, $\mu < 8d_{old} + 16$, and there have been at most $d_{old}(d_{old} + 1)$ joins in d_{old} steps, so $\mu < 8d_{old} + 16 + \frac{d_{old}(d_{old}+1)}{2^{d_{old}}} < 8d_{old} + 18$ before merging, and $\mu < 16d_{old} + 36$ afterwards. The maximum node has less than $21d_{new} + 61$ peers.

We now show that $\mu \geq 8d_{new} + 16$ for the next $d_{new} + 1$ phases after a reduction. At time $t - d_{old} - 1$, $\mu \geq 8d_{old} + 16 = 8d_{new} + 24$. The reduction doubles the average number of peers per node, hence $\mu \geq 16d_{new} + 48$. Further, there are at most $(d_{old} + 1)(d_{old} + 1) + (d_{new} + 1)(d_{new} + 1) = 2d_{new}^2 + 6d_{new} + 5$ leaves in the meantime, hence $\mu \geq 16d_{new} + 48 - \frac{2d_{new}^2+6d_{new}+5}{2^{d_{new}}} > 16d_{new} + 41 > 8d_{new} + 16$.

Finally, $\mu \leq 40d_{new} + 80$ for $d_{new} + 1$ phases. At time $t - d_{old}$, $\mu < 8d_{new} + 24$, so $\mu < 16d_{new} + 48$ after the reduction. There are at most $d_{old}(d_{old}+1) + (d_{new}+1)(d_{new}+1) = 2d_{new}^2 + 5d_{new} + 3$ joins, therefore $\mu < 16d_{new} + 48 + \frac{2d_{new}^2+5d_{new}+3}{2^{d_{new}}} < 16d_{new} + 54 < 40d_{new} + 80$.

Case $\mu > 40d + 80$*:* At time $t - d_{old}$, $\mu > 40d_{old} + 80 = 40d_{new} + 40$, so $\mu > 20d_{new} + 20$ after splitting; there are at most $d_{old}(d_{old} + 1) = d_{new}^2 - d_{new}$ leaves in d_{old} steps, so $\mu > 20d_{new} + 20 - \frac{d_{new}^2-d_{new}}{2^{d_{new}}} > 20d_{new} + 19$. According to Lemma 3, the minimum node has more than $15d_{new} + 15$ peers after splitting. At time $t - d_{old} - 1$, $\mu \leq 40d_{old} + 80$, and there are at most $(d_{old}+1)(d_{old}+1) = d_{old}^2 + 2d_{old} + 1$ joins. Hence, before splitting, $\mu \leq 40d_{old} + 80 + \frac{d_{old}^2+2d_{old}+1}{2^{d_{old}}} < 40d_{old} + 82$, and the maximum node has at most $45d_{old} + 86$ peers.

Next, we show that $\mu \geq 8d_{new} + 16$ for the $d_{new} + 1$ phases after the expansion. At time $t - d_{old}$, $\mu > 40d_{old} + 80 = 40d_{new} + 40$, hence $\mu > 20d_{new} + 20$ after the expansion. Moreover, there are at most $d_{old}(d_{old}+1) + (d_{new}+1)(d_{new}+1) = 2d_{new}^2 + d_{new} + 1$ leaves, and $\mu > 20d_{new} + 20 - \frac{2d_{new}^2+d_{new}+1}{2^{d_{new}}} > 20d_{new} + 17 \geq 8d_{new} + 16$. Finally, $\mu \leq 40d_{new} + 80$ for the next $d_{new} + 1$ steps: at time $t - d_{old} - 1$, $\mu \leq 40d_{old} + 80 = 40d_{new} + 40$, so $\mu \leq 20d_{new} + 20$ after the expansion; moreover, there are at most $(d_{old} + 1)(d_{old} + 1) + (d_{new} + 1)(d_{new} + 1) = 2d_{new}^2 + 2d_{new} + 1$ joins, thus $\mu \leq 20d_{new} + 20 + \frac{2d_{new}^2+2d_{new}+1}{2^{d_{new}}} < 20d_{new} + 24 < 40d_{new} + 80$.

In reality, the repairing operations will run *concurrently* to the adversary. However, as all operations are based on the state of Round 1, a phase can be considered as running uninterruptedly, that is, as if the adversary inserted $d + 1$ and removed $d + 1$ peers only *between* the phases. Thus, the properties shown above remain valid. However, we additionally have to postulate that there is always at least one *core peer*. We know that it is always possible to select $2d + 3$ core peers in Round 5 with respect to the state of Round 1. These peers have to survive until Round 6 of the next phase, so for twelve normal rounds in total; however, as the adversary $\mathcal{A}_{adv}(d+1, d+1, 6)$ removes at most $2d + 2$ peers in twelve rounds, this clearly holds.

Finally, we show that there are indeed enough peripheral peers in Round 3 such that core peers do not have to change the node for the peer distribution, that is: in Round 3, it holds that $|\mathcal{P}_v| > \frac{size(v)-size(u)}{2}$. We know that $size(v) \geq 3d + 10$ and $size(u) \geq 3d + 10$. As v has at most $2d + 3$ core peers, we have $|\mathcal{P}_v| \geq size(v) - (2d + 3) \geq size(v) - size(u) > \frac{size(v)-size(u)}{2}$. □

Finally, observe that it is possible to replace the complete bipartite graphs between adjacent hypercube nodes by bipartite matchings, reducing the peer degree from $O(\log^2 n)$ to $O(\log n)$. Apart from the lower degrees, all our results still hold up to constant factors.

4 The dynamic pancake system

The previous section has presented algorithms to maintain a hypercube topology with diameter $O(\log n)$ and where each peer has at most $O(\log n)$ neighbors. The topology is resilient to $O(\log n)$ worst-case changes per time unit which is asymptotically optimal. In this section we sketch how our construction can be adapted for another interesting network topology, namely for the *pancake graph* of order d defined in Definition 1.

The pancake graph and the unsolved problem of computing its diameter (e.g., [58]) was introduced in [13]. In terms of the group-theoretic model for network topologies introduced by Akers and Krishnamurthy [2], the pancake is an instance of a hierarchical *Cayley graph* [4]. We are not aware of any literature on dynamic token distribution or information aggregation on pancake graphs. The pancake graph has the interesting property that it is a graph with minimal maximum of peer degree and network diameter, namely $O(\log n/\log\log n)$ (see Fig. 2).

Besides the interesting diameter/degree trade-off (which is also achieved by other (Cayley) graphs), a major reason for us to study the pancake graph is that it is considered a difficult graph compared to a hypercube for example. Indeed, as we will see in the following, simulating the pancake in such a way that the network still has $O(\log n/\log\log n)$ diameter and degree requires a slightly different peer organization within a node. However, we are still able to devise token distribution and information aggregation algorithms on the pancake that can be translated into the simulated systems, providing evidence that our techniques work for a large class of graphs (beyond hypercubic topologies).

Definition 1 A *pancake graph* of order d is a graph $P_d = (V, E)$, with $V(P_d) = \{l_1 l_2 \ldots l_d \,|\, l_i \in \{1, \ldots, d\}, \forall i \neq$

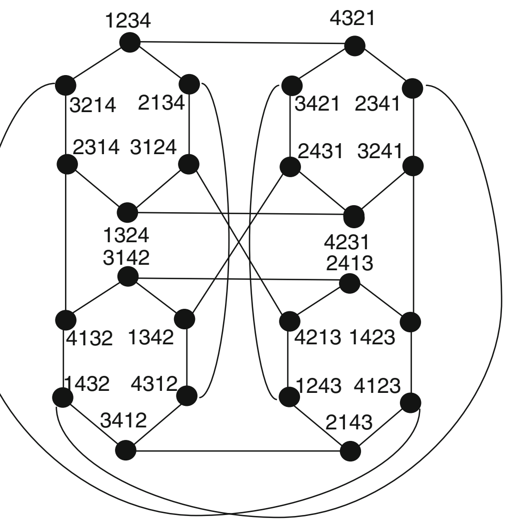

Fig. 2 A pancake graph of order 4 (P_4)

$j : l_i \neq l_j\}$, i.e., $V(P_d)$ is the set of all permutations on the set $[1, d]$. Let ρ_i denote a prefix-inversion of length i: $\rho_i(l_1 \dots l_i \dots l_d) = l_i l_{i-1} \dots l_1 l_{i+1} \dots l_d$. For $u, v \in V(P_d)$, it holds that $\{u, v\} \in E(P_d) \Leftrightarrow v = \rho_i(u)$ for $i \in \{2, \dots, d\}$. P_d is a $(d-1)$-regular graph of diameter smaller than $2d$ (by the straight-forward routing algorithm which adjusts one position after the other in two rounds each, starting from the least significant position).

Henceforth, we will refer to the set $\{a, a+1, \dots, b-1, b\}$ as $[a, b]$. Moreover, the number l_i at the i^{th} position of a node with label $v = l_1 \dots l_d$ will be called the ith *entry*.

Again, we *simulate* the pancake graph in our p2p system: each peer is part of a distinct pancake node, and each pancake node consists of $O(d^2)$ peers. A data item is redundantly stored by the peers of the node to which its identifier hashes. Peers have connections to other peers of their pancake node; additionally, some peers of neighboring pancake nodes are connected to each other. In case of joins or leaves, some of the peers have to change to another pancake node such that up to constant factors, all pancake 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 order of the pancake is increased or decreased by one, respectively.

The balancing of peers among the pancake nodes is again done by a dynamic token distribution algorithm. Moreover, we have a distributed information aggregation algorithm which estimates the number of peers in the system and adapts the order accordingly. Based on the described structure, we get a p2p system with peer degree and network diameter $O(\log n/\log\log n)$, implying time complexity $O(\log n/\log\log n)$ for the usual operations such as search. At the same time, our system tolerates $\Theta(\log n/\log\log n)$ worst-case joins and/or crashes per constant time interval.

4.1 Dimension change

The order of the pancake graph is changed according to the total number of peers in the system. For the expansion, node $l_1 \dots l_d \in V(P_d)$ splits into $d+1$ new nodes $\{(d+1)l_1 l_2 \dots l_d, l_1(d+1)l_2 \dots l_d, \dots, l_1 l_2 \dots l_d(d+1)\}$ of P_{d+1}, and vice versa for the reduction.

To be useful for our application, the pancake's order change from d_{old} to d_{new} has to fulfill a requirement: a node in $P_{d_{new}}$ must be able to compute its new neighbors *locally*, i.e., based on the information about the neighbors in $P_{d_{old}}$. We will now describe the expansion and the reduction of the order in turn and show that this criterion is indeed met in both cases.

4.1.1 Expansion

If the total number of peers in the system exceeds a certain threshold, each node $v = l_1 \dots l_d \in V(P_d)$ splits into $d+1$ new nodes $\{v^{exp}_{(1)} = (d+1)l_1 l_2 \dots l_d, v^{exp}_{(2)} = l_1(d+1)l_2 \dots l_d, \dots, v^{exp}_{(d+1)} = l_1 l_2 \dots l_d(d+1)\}$ of P_{d+1}. The following lemma states that the new neighbors of a node $v^{exp}_{(i)} \in V(P_{d+1})$ can easily be computed given the knowledge about the neighbors of the original node $v \in V(P_d)$.

Lemma 6 *Consider two arbitrary nodes u and v of P_d. It holds that if $\{u^{exp}_{(i)}, v^{exp}_{(j)}\} \in E(P_{d+1})$ for some $i, j \in \{1, \dots, d+1\}$, then $\{u, v\} \in E(P_d)$ or $u = v$.*

Proof If $\{u^{exp}_{(i)}, v^{exp}_{(j)}\} \in E(P_{d+1})$ there is a $k \in \{2, \dots, d+1\}$ such that $u^{exp}_{(i)} = \rho_k(v^{exp}_{(j)})$. If $d+1$ appears among the first k entries of $u^{exp}_{(i)}$ (and thus also of $v^{exp}_{(i)}$), the original nodes—having no entry $(d+1)$—are related by a prefix-inversion of length $k-1$: $u = \rho_{k-1}(v)$. If on the other hand the entry $(d+1)$ appears among the remaining entries, u and v are related by the same prefix-inversion: $u = \rho_k(v)$. □

4.1.2 Reduction

If the total number of peers in the system falls below a certain threshold, all nodes $l_1 \dots l_i(d+1)l_{i+1} \dots l_d \in V(P_{d+1})$ for $i \in [0, d]$ merge into a single node $l_1 \dots l_d \in V(P_d)$. Unfortunately, we cannot reverse the expansion directly. Instead, the reduction works as follows. First, the following DOMINATING SET [17] on P_{d+1} is computed: every node $v = l_1 \dots l_{d+1}$ having $l_1 = d+1$ becomes a dominator. We will call a dominator plus its adjacent (dominated) nodes a *cluster*. In the following, let $v^{dom}_{(1)} = (d+1)l_1 \dots l_d$ be a dominator and $v^{dom}_{(i)} = \rho_i(v^{dom}_{(1)}) = l_{i-1}l_{i-2} \dots (d+1)l_i \dots l_d$ its neighbor with prefix-inversion of length i, for $i \in [1, d+1]$. The idea is to contract each cluster with dominator $v^{dom}_{(1)} = (d+1)l_1 \dots l_d$ to a single node $v = l_1 \dots l_d \in V(P_d)$. Mind, however, that our clusters do not yield the desired reduction yet: in order to get the inverse operation of the expansion, each cluster has to exchange one dominated node with each of its adjacent clusters.

Before we explain the exchange of the dominated nodes in detail, we first prove that the set of nodes having $l_1 = d+1$ indeed forms a dominating set, that every dominated node is adjacent to exactly one dominator, and that dominators are independent.

Lemma 7 *Consider the graph P_{d+1}. The $d!$ nodes of P_{d+1} with first entry $l_1 = d+1$ build a dominating set, i.e., each node is either a dominator itself or adjacent to a dominator. Moreover, clusters are disjoint.*

Proof Consider an arbitrary node $v = l_1 l_2 \ldots l_{d+1}$. Assume that $l_i = d+1$ for some $i \in \{1, \ldots, d+1\}$. If $i = 1$, v is a dominator itself. Two nodes having $l_1 = d+1$ cannot be adjacent because of the prefix-inversion changes the first entry. If $i \neq 1$, there is exactly one neighbor of v which is a dominator, namely node $u = \rho_i(v)$. □

According to Lemma 7, each node belongs to exactly one cluster, hence the contraction operation is well-defined. However, as already mentioned, we additionally need to exchange dominated nodes between adjacent clusters. This is done as follows: the cluster with dominator $v^{dom}_{(1)} = (d+1)l_1 \ldots l_d$ sends its dominated node $v^{dom}_{(i+1)}$ to the cluster with dominator $(d+1)\rho_i(l_1 \ldots l_d)$, for $i \in [2, d]$.

It holds that after the exchange of the dominated nodes, (i) each cluster with dominator $v^{dom}_{(1)} = v^{exp}_{(1)} = (d+1)l_1 \ldots l_d$ which will contract to node $v = l_1 \ldots l_d$ consists of the nodes $v^{exp}_{(1)} = (d+1)l_1 \ldots l_d, v^{exp}_{(2)} = l_1(d+1) \ldots l_d, \ldots, v^{exp}_{(d+1)} = l_1 \ldots l_d(d+1)$, and (ii) the dominated node $v^{exp}_{(i)}$ for $i \in [3, d+1]$—before being transferred to the cluster dominated by $v^{dom}_{(1)}$—belonged to the cluster that will form the new node $\rho_i(v)$. To see this, note that node $v^{dom}_{(i)}$ is replaced by $\rho_{i-1}(v^{dom}_{(i)}) = \rho_{i-1}(l_{i-1} \ldots l_1(d+1)l_i \ldots l_d) = v^{exp}_{(i)}$, and that before the transfer, $v^{exp}_{(i)}$ belonged to the cluster dominated by $\rho_i(v^{exp}_{(i)}) = (d+1)l_{i-1} \ldots l_1 l_i \ldots l_d$ which will reduce to node $\rho_{i-1}(v)$. Thus, after the exchange, the following lemma holds.

Lemma 8 *The cluster contracting to node v consists of those nodes which v would also expand to, and the cluster has information about each of v's neighbors.*

4.2 Information aggregation

We now describe our algorithm $\mathcal{A}_{IA}$ which allows us to count the total number of peers in the pancake's nodes. Let $P_i(v)$ denote the sub-graph of the pancake graph P_d consisting of those nodes which share a postfix of length $d-i$ with a given node v. (Note that the graph induced by $P_i(v)$ is a pancake graph of order i.) The algorithm runs in $d-1$ phases and accumulates the total number of tokens in sub-graphs of increasing size.

Each phase consists of two rounds. In the first round of phase i, a node v sends the total number of tokens in its sub-graph $P_i(v)$—which is known by induction—to its neighbor $\rho_{i+1}(v)$. Thus, since prefix-inversion is a symmetric operation, v receives the total number of tokens in the sub-graph $P_i(\rho_{i+1}(v))$ from node $\rho_{i+1}(v)$. In the second round, node v sends this information to all neighbors $\rho_j(v)$ for $j < i+1$. Given the information about all $P_i(\rho_{i+1}(\rho_j(v)))$ (for $j < i+1$), the total number of tokens in the sub-graph $P_{i+1}(v)$ can be computed: $\tau(P_{i+1}(v)) = \tau(P_i(v)) + \sum_{j=1}^{i} \tau(P_i(\rho_{i+1}(\rho_j(v))))$, where $\tau(\cdot)$ denotes the number of tokens in the corresponding sub-graph. Hence, by induction, after $d-1$ phases, every node can compute the total number of tokens in the system.

Theorem 6 *$\mathcal{A}_{IA}$ provides all nodes with the correct total number of tokens in the system after $d-1$ phases.*

Proof By induction over the phases we show that after phase i, it holds that each node v knows the total number of tokens in $P_{i+1}(v)$.

$i = 0$: Before the first phase, a node v only knows its own tokens, and as there is only one node in $P_1(v)$, the claim holds trivially.

$i \to i+1$: By the induction hypothesis, after phase i, each node $v = l_1 \ldots l_d$ knows the total number of tokens in the sub-graph $P_{i+1}(v)$. In phase $i+1$, node v learns the total number of tokens in the sub-graphs $P_{i+1}(\rho_{i+2}(\rho_j(v)))$ for $j < i+2$. This facilitates the computation of the total number of tokens in $P_{i+2}(v)$.

Note that the nodes $\rho_j(v)$ for $j < i+2$ all have a different first entry and share the postfix $l_{i+2}l_{i+3} \ldots l_d$ with v. Performing a ρ_{i+2} prefix-inversion yields a member for each sub-graph with postfix $l_{i+3}l_{i+4} \ldots l_d$ of length $d-(i+2)$. Therefore, combining the information of the sub-graphs gives the total number of tokens in $P_{i+2}(v)$. □

$\mathcal{A}_{IA}$ is executed all the time and in a pipelined fashion, i.e., all phases run concurrently. This way, all nodes always get a consistent result even if the adversary concurrently adds and removes tokens (peers). Moreover, the result always corresponds to the exact state of the system $d-1$ phases ago.

4.3 Token distribution

Our goal is again to minimize the maximum difference of the number of tokens of any two pancake nodes, denoted by the *discrepancy* ϕ. Analogous to the information aggregation algorithm, our token distribution algorithm $\mathcal{A}_{TD}$ exploits the recursive structure of the pancake graph. In a first step, all pancakes of order 2 balance their tokens. Then, the pancakes of order $3, 4, \ldots$ exchange tokens. Pancakes of order i can thereby build on the fact that all pancakes of order $i-1$ have balanced the token levels of their nodes. A detailed description of $\mathcal{A}_{TD}$ is given in Algorithm 1. We assume that we have a dominating set for each pancake $P_i(v)$. For example, the dominators could again be all nodes of $P_i(v)$ having the largest of the first i entries at the first position. Note that, by definition, entries $i+1$ to d are fixed for all nodes of $P_i(v)$.

Algorithm 1 Token Distribution $\mathcal{A}_{TD}$ (node v)

```
1: for $i := 2$ to $d$ do
2:     send all tokens to $\rho_i(v)$;
3:     send all tokens to dominator in $P_i(v)$;
4:     dominators send tokens to nodes of their clusters;
5: end for
```

Let $P_i(v)$ be the pancake of order i. After the i^{th} iteration of $\mathcal{A}_{TD}$, for all v, all nodes of $P_i(v)$ have the same number of tokens. Hence, at the end ($i=d$) all nodes of the pancake have the same number of tokens. In Line 4 of $\mathcal{A}_{TD}$, it is not specified how many tokens to send to which nodes if the number of tokens at a node is not divisible by i. There is also no explicit notion of tokens which are added or removed by an adversary during the algorithm. In the following, we will prove that the algorithm perfectly distributes tokens if tokens are fractional, that is, if they can be divided arbitrarily and if no tokens are added or removed during the algorithm (static token distribution). We will then analyze the effects of adversarial insertions and deletions and of integer tokens.

Lemma 9 *$\mathcal{A}_{TD}$ perfectly solves the static fractional token distribution problem on a pancake of order d.*

Proof As outlined above, we prove the lemma by induction over i. Since $P_1(v)$ is a single node, clearly at the beginning all nodes of $P_1(v)$ have the same number of tokens. Let us therefore assume that for all nodes u, each node of $P_{i-1}(u)$ has the same number of tokens $\tau_{i-1}(u)$. The pancakes $P_{i-1}(u)$ of order $i-1$ belonging to $P_i(v)$ can be characterized by their ith entry. Let l_i be the ith entry of the nodes of $P_{i-1}(u)$. In Line 2 of $\mathcal{A}_{TD}$, a node u of $P_{i-1}(u)$ moves all tokens to $\rho_i(u)$, that is, all tokens are moved to a node with l_i as its first entry. Hence, after Line 2, all nodes of $P_i(u)$ with first entry l_i have $\tau_{i-1}(u)$ tokens.

In Lines 3 and 4, each cluster (dominator plus neighbors) distributes all its tokens equally among the members of the cluster. It therefore remains to show that each cluster of $P_i(u)$ has the same number of tokens. However, since in each cluster, every possible first entry occurs exactly once, this is clear from the discussion of the first step of the algorithm (Line 2). □

We will now show how dynamic insertions and deletions of tokens affect the fractional token distribution of $\mathcal{A}_{TD}$. For the dynamic token distribution algorithm, we assume that the $d-1$ iterations of the algorithm are repeated, that is, after $i=d$, we start again at $i=2$.

Lemma 10 *If in every iteration of $\mathcal{A}_{TD}$ at most J tokens are added and at most L tokens are removed, the algorithm guarantees that at all times $t \geq d-1$, the maximal difference between the numbers of fractional tokens between any two nodes is $3(J+L)$.*

Proof To start, we only consider insertions and neglect deletions. Because all operations of the algorithm are linear in the sense that they can be combined independently, we can look at each token independently. By Lemma 9, each token which is added before the first iteration of the algorithm is distributed equally among $i! \geq 2^i$ nodes after iteration i. A token which is added after iteration j is distributed among $i!/j! \geq 2^{i-j}$ nodes after iteration i. All tokens which were inserted before the last complete execution of $\mathcal{A}_{TD}$ are equally distributed among all nodes of the pancake. We therefore only have to look at the last complete execution and at the current execution of the algorithm. All tokens which are inserted in the current execution of $\mathcal{A}_{TD}$ are distributed among at least 2^t nodes, t iterations after the insertion. Therefore, by a geometric series argument, there are at most $2J$ tokens per node which were inserted in the current iteration. All tokens which were inserted before the end of the last complete execution of the algorithm, were distributed among at least d nodes after the last complete execution. Since in iteration i, each node distributes its tokens among i different nodes and each node receives tokens from i different nodes, all the tokens from the last complete execution of the algorithm remain distributed among at least d nodes. Because there are at most $(d-1)J$ such tokens, each node has less than one of them. Together, the difference between the number of tokens at the heaviest and the lightest node becomes $3J$. For deleted tokens the same argumentation as for inserted tokens holds. □

We have analyzed the token distribution algorithm for the idealized case where tokens can be divided arbitrarily. In our application, tokens correspond to peers, and we again have to extend the analysis to integer tokens. We assume that in Line 4, tokens are distributed as equally as possible. That is, if there are k tokens in a cluster, some of the nodes receive $\lfloor k/i \rfloor$ tokens and some nodes receive $\lceil k/i \rceil$ tokens.

Lemma 11 *The (absolute) difference between the number of integer tokens and the number of fractional tokens at any node is always upper bounded by $2d$.*

Proof We start the proof by looking at iteration i of $\mathcal{A}_{TD}$. Assume that before iteration i, the difference between the number of integer tokens and the number of fractional tokens is at most ξ at each node. If there are token insertions or deletions at a node, this difference does not change because insertions and deletions affect the numbers of fractional and integer tokens in the same way. In Line 2, all tokens are moved and therefore ξ remains unchanged. In Lines 3 and 4, tokens are distributed equally among i nodes of a cluster. If there are k tokens in such a cluster, each node gets between $\lfloor k/i \rfloor$ and $\lceil k/i \rceil$ tokens. If every node got exactly k/i tokens, the difference between fractional and integer would remain at most ξ. Due to the rounding, the difference can therefore grow to at most $\xi + 1$ after iteration i. Hence, after t iterations, the absolute difference between the numbers of fractional and integer tokens is at most t.

To prove that at each node, the number of integer tokens cannot deviate from the number of fractional tokens by more than $2d$, we need the following observation. By Lemma 9, fractional tokens are distributed equally among all nodes after their first complete execution of $\mathcal{A}_{TD}$, that is, after less than $2d$ iterations. Therefore, the number of fractional tokens at each node only depends on the insertions and deletions of the last $2d$ iterations and on the total number of tokens in the system. Therefore, the distribution of fractional tokens is the same if we assume that before the last $2d$ iterations, the number of fractional tokens at each node was equal to the number of integer tokens. By the above argumentation, the difference between the numbers of integer and fractional tokens at a node can have grown to at most $2d$ in those $2d$ iterations. □

By combining Lemmas 9, 10, and 11, we obtain the following theorem about the dynamic integer token distribution algorithm.

Theorem 7 *The discrepancy ϕ of the dynamic integer token distribution algorithm is at most $\phi \leq 4d + 3(J + L)$.*

The algorithm $\mathcal{A}_{TD}$ is formulated in the form which makes the proofs of this section as simple as possible. It is of course not desirable that all nodes first have to move all tokens to dominator nodes which then redistribute the tokens. Especially in the case where no insertions or deletions occur, we would like the system to stabilize to a point where no tokens have to be moved around. It is not difficult to implement $\mathcal{A}_{TD}$ in a way which has this property. In Line 2, two nodes u and $\rho_i(u)$ exchange all their tokens. They can of course obtain the same effect by computing the difference between the number of tokens and by only moving this number of tokens in the appropriate direction. A similar trick can be applied for Lines 3 and 4. The dominator nodes can collect all the necessary information and decide about the necessary movements of tokens.

4.4 Node representation

The algorithms so far have all been described on the level of pancake graphs. In this section, we take a more detailed look at the internals of the system. We first present the representation of the pancake's nodes and edges and then give an algorithm which allows to maintain these structures against a concurrent adversary. We omit the peer-level description of some components, for example the token distribution or the information aggregation algorithm. These operations are straight-forward and can be done with similar techniques.

4.4.1 The grid

The peers of a node $v \in V(P_d)$ are arranged to form a 2-dimensional grid G_v consisting of exactly $d+1$ columns, while the number of rows R may vary depending on the total number of peers in v.

Let $\tau(v)$ be the total number of peers in node v and let $R = \lfloor \tau(v)/(d+1) \rfloor$. The first $R \cdot (d+1)$ peers are arranged in a 2-dimensional grid with $d+1$ columns and R complete rows, such that every peer occupies exactly one position $G_v[x, y]$ for $x \in [0, d]$ and $y \in [0, R-1]$. The remaining $\tau(v)$ *mod* $(d+1)$ peers—from now on called *extra peers*—are located in an incomplete additional row $G_v[i, R]$ for $i \in [0, \tau(v)$ *mod* $(d+1)]$. Inside a row or column, the peers are completely connected ("intra-connections"): a peer at $G_v[x, y]$ is connected to the peers $G_v[x, i]$ for $i \in [0, R]$ and $G_v[i, y]$ for $i \in [0, d]$. As the extra peers do not form a complete row, they are more vulnerable; thus, they additionally participate in row $R-1$, i.e., we also have connections between $G_v[i, R]$ for $i \in [0, d+1]$ and *all* peers $G_v[j, R-1]$ for $j \in [0, \tau(v)$ *mod* $(d+1)]$.

Additionally, we need to specify the representation of the pancake's edges ("inter-connections"). The idea is as follows: if two nodes u and v are connected in the pancake graph P_d, i.e., $\{u, v\} \in E(P_d)$, then each peer $G_u[i, 0]$ is connected to the peer occupying $G_v[i, 0]$, for $i \in [0, d]$. In the following, we will call the peers in the lowest row (row 0) the *core* of the corresponding node. Thus, two nodes are connected by a *matching* between their cores. The representation of the pancake's nodes is depicted in Fig. 3.

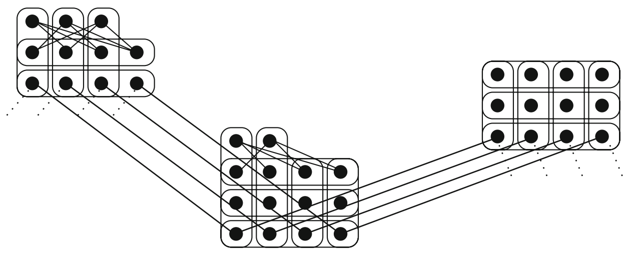

Fig. 3 The peers of a pancake node are arranged as a grid with $d+1$ columns. A peer has connections to all peers in its row plus to all peers in its column. The pancake's edges are represented by a matching between the peers of the *bottom row*

4.4.2 Grid maintenance

Algorithm $\mathcal{A}_{GRID}$ needs several rounds. The main theme is similar to the one proposed for the hypercube graph: at the beginning, a snapshot of the state (living peers, etc.) of the system is made. The following rounds are then solely based on this information—ignoring the fact that some peers have been crashed by the concurrent adversary in the meantime. That is, by using sufficient redundancy, we do not have to take the crashed and newly joined peers into consideration until the maintenance algorithm restarts with the first round.

$\mathcal{A}_{GRID}$ consists of two phases. In the first phase, the following information is broadcast throughout the grid: (1) the positions where peers have left, (2) the IP addresses of the peers that have joined, (3) the IP addresses of the extra peers, and (4) the IP addresses of the peers in row $R-1$. The second phase is based on this information and works as follows: every surviving peer can locally compute which peers will take the positions of the peers that left (gaps in the grid). Thereby, newly joined peers are taken into account first, and if this is not enough, the extra peers are used. If there are still gaps in the grid, the peers of the top row are used, and if necessary, the number of rows is decremented ($R := R-1$). If on the other hand there are still joining peers left after all gaps have been filled, these peers are added to the top row, creating a new top row if necessary ($R := R+1$). After this local computation, the peers that have to fill the gaps are provided with the information about their new neighbors. We can guarantee that no row is removed completely and that there is always a complete column in the presence of a concurrent adversary $\mathcal{A}_{ADV}(d/2, d/2, 5)$ who adds and removes at most $d/2$ peers in any time period of 5 rounds. Moreover, also the pancake's edges are repaired in constant time since we ensure that two adjacent pancake nodes always have at least two living adjacent core peers which can reestablish the matching between the cores.

We now give the detailed description of $\mathcal{A}_{GRID}$. We write $G_v[\cdot, y]$ and $G_v[x, \cdot]$ to denote all (surviving) peers in the y^{th} row and in the x^{th} column respectively. In the following, we assume the extra peers to participate in both rows R *and* $R-1$, i.e., they send and receive messages for both rows.

Round 1: The snapshot is made: a surviving peer at position $G_v[x, y]$ sends its IP address and the IP addresses of its joiners to all peers in $G_v[\cdot, y]$.

Round 2: Each peer at position $G_v[x, y]$ sends the addresses of its joiners plus the information in which column of its row peers have left to $G_v[x, \cdot]$.

Round 3: Each peer at position $G_v[x, y]$ forwards the information received in Round 2 to the peers $G_v[\cdot, y]$.

Round 4: Now the new form of G_v is computed locally: if a peer at $G_v[x, y]$ has missing neighbors on its row or column, it computes which joiner or—if necessary—which extra peer or which peer in the top row has to replace it. If there are enough new peers, the number of rows is incremented, and vice versa if more than all extra peers are used for repairing. Each peer having a missing neighbor on its row sends the information about all neighbors of this row or column directly to the peer which will replace it. Additionally, the information required to establish the top rows is provided to the responsible peers. Finally, in order to repair the matching between adjacent nodes, the peers of the old core which are still alive send the addresses of the new core peers to the old neighboring cores.

Round 5: The old core broadcasts the new partners of the matchings within $G_v[\cdot, 0]$; this ends the repairing operation with respect to the snapshot's state.

4.4.3 Expansion

When the pancake graph's order is incremented from d to $d+1$, each node v must split into $d+1$ new nodes. Since the grid G_v consists of $d+1$ columns, there is a simple way to perform the expansion on the grid level: every column becomes one new node.

According to Sect. 4.1, two neighboring expanded nodes have already been adjacent in P_d (or originate from the same node). Assume that two columns, one in G_v and the other one in G_u, for two expanding adjacent nodes $u, v \in V(P_d)$, become neighbors in P_{d+1}. With the grid as described so far, these two columns have only one connection to each other (one pair of core peers). In order to increase the fault-tolerance the following mechanism is applied: as soon as there are enough peers in the system and there are at least $d+2$ complete rows in each node, adjacent nodes $u, v \in V(P_d)$ start to establish a matching between the columns in G_u and G_v which will become neighbors if the graph is expanded. In order to limit the information that is sent, we establish this matching stepwise, ensuring that it is finished before the node actually has to split. This is done in $d+1$ phases, in phase i for the matching to neighbor $\rho_i(v)$. The idea is that each peer at $G_v[x, y]$ with $y \in [1, d+2]$ sends its IP address to the peer $G_v[y-1, 0]$. Peer $G_v[y-1, 0]$ is then responsible to transfer the y^{th} row to the corresponding peers $G_{\rho_i(v)}[y, 0]$ for $i \in [2, d+2]$. From there, the information is broadcast to $G_{\rho_i(v)}[\cdot, y]$. This mechanism guarantees that between two neighboring columns, at least one connection will be finished, even in the presence of a concurrent adversary. Once the matching is established, it is maintained as long as there are at least $d+2$ rows.

The expansion then works as follows. We consider a node $v = l_1 \ldots l_d$ with grid G_v. The column $G_v[i, \cdot]$ for $i \in [1, d+$

1] will form the new node $v_{(i)}^{exp} = l_1 \dots l_{i-1}(d+1)l_i \dots l_d$. Since peers of the i^{th} column $G_v[i, \cdot]$ are completely connected, the expansion can be performed in two rounds: it is straight-forward to locally compute the form of the new grids $G_{v_{(i)}^{exp}}$, including cores and inter-connections, and send this information to nodes $\rho_j(v_{(i)}^{exp})$ for $j \in [2, d+1]$.

Round 1: The peers of the ith column $G_v[i, \cdot]$ which will form the new node $v_{(i)}^{exp}$ are completely connected, and each peer in $v_{(i)}^{exp}$ can locally compute the form of $G_{v_{(i)}^{exp}}$. The information about the new core is sent to nodes $\rho_j(v_{(i)}^{exp})$ for $j \in [2, d+1]$, using the connections of the matching.

Round 2: The peers in $v_{(i)}^{exp}$ send the information about the neighboring cores received in Round 1 to their own new core.

4.4.4 Reduction

The reduction of the pancake's order is more elaborate: reducing the order from $d+1$ to d requires $d+1$ grids to merge into one. Additionally, some peers are bound to change nodes (cf Sect. 4.1).

Similarly to the notation introduced in Sect. 4.1, let $v_{(1)}^{dom} \in V(P_{d+1})$ be the dominator of a cluster that contracts to $v \in V(P_d)$ and let $v_{(i)}^{dom} = \rho_i(v_{(1)}^{dom})$. To reduce the order of the pancake graph, we must exchange the nodes $v_{(i+1)}^{dom}$ with $u_{(i+1)}^{dom}$ for $i \in [2, d]$ where $u = \rho_i(v)$, and then merge the clusters into one node v (cf Sect. 4.1).

On the grid level, a constant number of rounds is needed for this order reduction. Basically, the procedure is as follows. First we turn $G_{v_{(i)}^{dom}}$ for $i \in [1, d+1]$ into a clique and the information about the core of $G_{v_{(1)}^{dom}}$ is sent to $\rho_{i-1}(v_{(i)}^{dom})$ (node exchange, cf Sect. 4.1). Now, the new grid of node v will be formed. For this, let again $v_{(i)}^{exp}$ for $i \in [1, d+1]$ be the nodes which will form v after the node exchange, $v_{(1)}^{exp}$ being the dominator. After $v_{(1)}^{exp}$ learned about its new dominated nodes, it sends *all* its peers' addresses to $v_{(i)}^{exp}$ for $i \in [2, d+1]$. With this information, a first version of G_v can be computed, where column i is given by $v_{(i)}^{exp}$. Based on this structure, the final grid can be obtained by a rearrangement.

4.5 The system

The n peers in our system are arranged in a simulated pancake topology of order d. The data of the DHT is stored as follows. Let $hash(\cdot)$ be a hash function which, given an identifier ID, outputs a random permutation on some set $[1, N]$, where N is a sufficiently large global integer constant. A data item with identifier ID is stored on the node $v \in V(P_d)$ which is determined by the ordering of the smallest d numbers of $hash(ID)$. A data item is not copied to *all* peers in that node, but only replicated on the core at the bottom row. Recall that this has the advantage that—if we use peers in topmost rows for the peer distribution—unnecessary copying of data can be avoided when peers move between nodes, while we are still able to tolerate the same powerful adversary. Finally, observe that routing is simple in the pancake system: assume that a peer in a node $u = l_1 l_2 \dots l_d$ wants to find a data item which hashes to a node $v = \widehat{l_1}\widehat{l_2} \dots \widehat{l_d}$. The lookup operation proceeds by correcting one "coordinate" at a time, starting at the back: from node $u = l_1 l_2 \dots l_d$ the request is forwarded to node $l_d \dots l_{j+1} l_1 l_2 \dots l_{j-1} \widehat{l_d}$, etc.

We now describe how to assemble the components to form a peer-to-peer system resilient to an adversary $\mathcal{A}_{ADV}(\Theta(\log n/\log\log n), \Theta(\log n/\log\log n), 1)$. We permanently run $\mathcal{A}_{IA}$ to estimate the total number of peers in the system and adapt the pancake's order accordingly, $\mathcal{A}_{TD}$ to distribute the peers evenly among the pancake's nodes, and $\mathcal{A}_{GRID}$ to maintain the grid. When the order of the pancake is changed, both $\mathcal{A}_{IA}$ and $\mathcal{A}_{TD}$ are restarted. This is possible because our system guarantees that after a change of the pancake's order, there are sufficiently many rounds without another order change such that the estimations of the total number of peers are up-to-date.

Taking into account that $\mathcal{A}_{IA}$ delivers the estimated number of peers with a delay of $d-1$ phases, and that according to Theorem 7, the difference between the total number of peers at any two nodes is bounded by $O(d)$ if there are $O(d)$ joins and leaves per time unit, we have the following theorem.

Theorem 8 *The pancake p2p system guarantees peer degree and network diameter* $O(d)$ *in the presence of an adversary who inserts and deletes* $\Theta(d)$ *peers per unit time. Each node always has at least one living core peer and no data is lost. Moreover, it holds that* $d = \Theta(\log n/\log\log n)$*, where* n *is the total number of peers in the system.*

The proof of Theorem 8 is similar to the deduction of Theorem 5, and is omitted here.

5 Conclusion

This article presented algorithms to maintain p2p networks under *worst-case* joins and leaves. It is often justified to study alternative churn models, e.g., *probabilistic* models [12] where peers join and leave according to a *Poisson process*. However, here we pursue a more conservative approach as this gives stronger guarantees. In addition, more optimistic models do not take attackers or viruses into account which exploit the p2p topology and propagate along the p2p system's links, indeed harming certain parts of the network more severely than others.

Two systems have been described which are optimal in the sense that there cannot exist topologies with a smaller peer degree which are robust to the same amount of churn. We believe that our techniques are applicable to many other graphs beyond the ones studied in this article. All that is needed is a token distribution and information aggregation algorithm on the graph. Indeed, the reason for studying the pancake graph in this article (in addition to the hypercube graph) was, besides its interesting properties, that it is known to be a difficult graph. However, it turned out that although some additional reasoning about how to simulate the topology by the peers is required, our approach also works there.

In practice, a simpler graph is typically advantageous, and hence, the hypercube system may be preferred over the pancake system. In particular, in future it would also be interesting to maintain alternative hypercubic structures similar to the ones used in Pastry, where there is no global dimension change but where the graph can evolve more locally. Indeed, based on the ideas presented here, we have developed *eQuus* [32], a DHT which connects peers in such a way that the peer degree and network diameter is always bounded by $O(\log n)$ with high probability, where n is the total number of peers in the network. eQuus gives less guarantees regarding the worst-case efficiency of the topology under churn. However, in contrast to the work presented here, it is *locality-aware* in the sense that communication takes place along routes whose latencies are close to optimal. eQuus has not been deployed yet and we were hence not able to collect empirical traces from this system's behavior in the wild. It also remains an open question whether locality-awareness can be achieved without sacrificing certain robustness properties.

A particularly important direction for future research is self-stabilization: How can our techniques be combined with the recent advances in topological self-stabilization in order to give recovery guarantees in times where the bounds on the adversarial power are exceeded?

Acknowledgments The authors would like to thank Joest Smit for interesting discussions on the pancake topology.

References

1. Abraham, I., Awerbuch, B., Azar, Y., Bartal, Y., Malkhi, D., Pavlov, E.: A generic scheme for building overlay networks in adversarial scenarios. In: Proceedings 17th International Symposium on Parallel and Distributed Processing (IPDPS) (2003)
2. Akers, S.B., Krishnamurthy, B.: A group-theoretic model for symmetric interconnection networks. IEEE Trans. Comput. **38**(4), 555–566 (1989)
3. Albrecht, K., Arnold, R., Gähwiler, M., Wattenhofer, R.: Aggregating information in peer-to-peer systems for improved join and leave. In: Proceedings 4th IEEE International Conference on Peer-to-Peer Computing (P2P) (2004)
4. Annexstein, F., Baumslag, M., Rosenberg, A.L.: Group action graphs and parallel architectures. SIAM J. Comput. **19**(3), 544–569 (1990)
5. Awerbuch, B., Scheideler, C.: The hyperring: a low-congestion deterministic data structure for distributed environments. In: Proceedings of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 318–327 (2004)
6. Awerbuch, B., Scheideler, C.: Robust random number generation for peer-to-peer systems. In: Proceedings of 10th International Conference on Principles of Distributed Systems (OPODIS) (2006)
7. Awerbuch, B., Scheideler, C.: Towards a scalable and robust DHT. In: Proceedings of 18th ACM Symposium on Parallel Algorithms and Architectures (SPAA) (2006)
8. Bhagwan, R., Savage, S., Voelker, G.: Understanding availability. In: Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS) (2003)
9. Bhargava, A., Kothapalli, K., Riley, C., Scheideler, C., Thober, M.: Pagoda: a dynamic overlay network for routing, data management, and multicasting. In: Proceedings of 16th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 170–179 (2004)
10. Clouser, T., Nesterenko, M., Scheideler, C.: Tiara: A self-stabilizing deterministic skip list. In: Proceedings of SSS (2008)
11. Cybenko, G.: Dynamic load balancing for distributed memory multiprocessors. J. Parallel Distrib. Comput. **7**, 279–301 (1989)
12. Datta, A., Aberer, K.: Internet-scale storage systems under churn. In: Proceedings of 6th IEEE International Conference on Peer-to-Peer Computing (P2P) (2006)
13. Dweighter, H. (a.k.a. J. E. Goodman): Problem E2569. American Mathematical Monthly, vol. 82 (1975)
14. Fiat, A., Saia, J.: Censorship resistant peer-to-peer content addressable networks. In: Proceedings of 13th Symposium on Discrete Algorithms (SODA) (2002)
15. Fiat, A., Saia, J., Young, M.: Making chord Robust to byzantine attacks. In: Proceedings of European Symposium on Algorithms (ESA) (2005)
16. Gall, D., Jacob, R., Richa, A., Scheideler, C., Schmid, S., Täubig, H.: Brief announcement: on the time complexity of distributed topological self-stabilization. In: Proceedings of International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS) (2009)
17. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of books in the mathematical sciences). W.H. Freeman, San Francisco (1979)
18. Godfrey, P.B., Shenker, S., Stoica, I.: Minimizing churn in distributed systems. In: Proceedings of ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (2006)
19. Guha, S., Daswani, N., Jain, R.: An experimental study of the skype peer-to-peer VoIP system. In: Proceedings of 5th International Workshop on Peer-to-Peer Systems (IPTPS) (2006)
20. Haeberlen, A., Mislove, A., Post, A., Druschel, P.: Fallacies in evaluating decentralized systems. In: Proceedings 5th International Workshop on Peer-to-Peer Systems (IPTPS) (2006)
21. Harvey, N., Munro, J.: Deterministic skipnet. Inf. Process. Lett. **90**(4), 205–208 (2004)
22. Jacob, R., Richa, A., Scheideler, C., Schmid, S., Täubig, H.: A distributed polylogarithmic time algorithm for self-stabilizing skip graphs. In: Proceedings of Annual ACM Symposium on Principles of Distributed Computing (PODC) (2009)
23. Jacob, R., Ritscher, S., Scheideler, C., Schmid, S.: A self-stabilizing and local delaunay graph construction. In: Proceedings of 20th International Symposium on Algorithms and Computation (ISAAC) (2009)
24. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proceedings of 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS) (2003)

25. Kothapalli, K., Scheideler, C.: Supervised peer-to-peer systems. In: Proceedings of International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN), pp. 188–193 (2005)
26. Kuhn, F., Locher, T., Wattenhofer, R.: Tight bounds for distributed selection. In: Proceedings of 19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA) (2007)
27. Kuhn, F., Schmid, S., Smit, J., Wattenhofer, R.: A blueprint for constructing peer-to-peer systems Robust to dynamic worst-case joins and leaves. In: Proceedings of 14th IEEE International Workshop on Quality of Service (IWQoS) (2006)
28. Kuhn, F., Schmid, S., Wattenhofer, R.: A self-repairing peer-to-peer system resilient to dynamic adversarial churn. In: Proceedings of 4th International Workshop on Peer-To-Peer Systems (IPTPS) (2005)
29. Li, J., Stribling, J., Morris, R., Kaashoek, M.F., Gil, T.M.: A performance versus cost framework for evaluating DHT design tradeoffs under churn. In: Proceedings of 24th Annual IEEE Conference on Computer Communications (INFOCOM) (2005)
30. Li, X., Misra, J., Plaxton, C.G.: Active and concurrent topology maintenance. In: Proceedings of 18th Annual Conference on Distributed Computing (DISC) (2004)
31. Liben-Nowell, D., Balakrishnan, H., Karger, D.: Analysis of the evolution of peer-to-peer systems. In: Proceedings of 21st Annual Symposium on Principles of Distributed Computing (PODC), pp. 233–242 (2002)
32. Locher, T., Schmid, S., Wattenhofer, R.: eQuus: a provably Robust and locality-aware peer-to-peer system. In: Proceedings of 6th IEEE International Conference on Peer-to-Peer Computing (P2P) (2006)
33. Mitra, B., Ghose, S., Ganguly, N.: Effect of dynamicity on peer-to-peer networks. In: Proceedings of 14th International Conference on High Performance Computing (HiPC) (2007)
34. Mosk-Aoyama, D., Shah, D.: Computing separable functions via gossip. In: Proceedings of 25th Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 113–122 (2006)
35. Naor, M., Wieder, U.: Novel architectures for P2P applications: the continuous-discrete approach. In: Proceedings of 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 50–59 (2003)
36. Pandurangan, G., Raghavan, P., Upfal, E.: Building low-diameter P2P networks. In: Proceedings of 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS) (2001)
37. Patt-Shamir, B.: A note on efficient aggregate queries in sensor networks. Theor. Comput. Sci. **370**(1–3), 254–264 (2007)
38. Peleg, D.: Distributed Computing A Locality-sensitive Approach. Society for Industrial and Applied Mathematics, Philadelphia, PA (2000)
39. Peleg, D., Upfal, E.: The token distribution problem. SIAM J. Comput. **18**(2), 229–243 (1989)
40. Plaxton, C.G.: Load Balancing, Selection and Sorting on the Hypercube. In: Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 64–73 (1989)
41. Qiao, Y., Bustamante, F.E.: Structured and unstructured overlays under the microscope—a measurement-based view of two P2P systems that people use. In: Proceedings of the USENIX Annual Technical Conference (2006)
42. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 161–172 (2001)
43. Rhea, S., Chun, B.-G., Kubiatowicz, J., Shenker, S.: Fixing the embarrassing slowness of openDHT on planetLab. In: Proceedings of the 2nd Conference on Real, Large Distributed Systems (WORLDS) (2005)
44. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a DHT. In: Proceedings of the USENIX Annual Technical Conference (2004)
45. Rowstron, A., Druschel, P.: Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pp. 329–350 (2001)
46. Saia, J., Fiat, A., Gribble, S., Karlin, A., Saroiu, S.: Dynamically fault-tolerant content addressable networks. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS) (2002)
47. Saroiu, S., Gummadi, P.K., Gribble, S.D.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of the Multimedia Computing and Networking (MMCN) (2002)
48. Scheideler, C.: How to spread adversarial nodes? rotate! In: Proceedings of the 37th ACM Symposium on Theory of Computing (STOC) (2005)
49. Scheideler, C., Schmid, S.: A distributed and oblivious heap. In: Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP) (2009)
50. Sen, S., Wang, J.: Analyzing peer-to-peer traffic across large networks. IEEE/ACM Trans. Netw. **12**(2), 219–232 (2003)
51. Shirazi, B.A., Kavi, K.M., Hurson, A.R.: Scheduling and Load Balancing in Parallel and Distributed Systems. IEEE Computer Science Press, Los Alamitos (1995)
52. Steiner, M., Biersack, E.W., Ennajjary, T.: Actively monitoring peers in KAD. In: Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS) (2007)
53. Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In: Proceedings of the 6th Internet Measurement Conference (IMC) (2006)
54. Tian, J., Dai, Y.: Understanding the dynamic of peer-to-peer systems. In: Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS) (2007)
55. Van Renesse, R., Birman, K.P., Vogels, W.: Astrolabe: a robust and scalable technology for distributed system monitoring, management, and data mining. ACM Trans. Comput. Syst. **21**(2), 164–206 (2003)
56. Van Renesse, R., Bozdog, A.: Willow: DHT, aggregation, and publish/subscribe in one protocol. In: Proceedings of the 3rd International Workshop on Peer-To-Peer Systems (IPTPS) (2004)
57. Yang, M., Chen, H., Zhao, B.Y., Dai, Y., Zhang, Z.: Deployment of a large scale peer-to-peer social network. In: Proceedings of the 1st Workshop on Real, Large Distributed Systems (2004)
58. Yusuke, K., Keiichi, K., Yuji, S.: Computing the diameter of the Pancake Graph. Joho Shori Gakkai Kenkyu Hokoku, **42** (2004)