A Blueprint for Constructing Peer-to-Peer Systems Robust to Dynamic Worst-Case Joins and Leaves

Fabian Kuhn
Microsoft Research
Silicon Valley
1065 La Avenida
Mountain View, CA, USA
kuhn@microsoft.com

Stefan Schmid
Computer Engineering and
Networks Laboratory (TIK)
ETH Zurich
8092 Zurich, Switzerland
schmiste@tik.ee.ethz.ch

Joest Smit
Computer Engineering and
Networks Laboratory (TIK)
ETH Zurich
8092 Zurich, Switzerland
jmsmit@ethz.ch

Roger Wattenhofer
Computer Engineering and
Networks Laboratory (TIK)
ETH Zurich
8092 Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

***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 paper, we devise techniques to build dynamic peer-to-peer systems which remain fully functional in spite of an adversary which continuously adds and removes peers. We exemplify our algorithms on a pancake topology and present a system which maintains peer degree and network diameter $\mathrm{O}(\frac{\log n}{\log\log n})$, where n is the total number of peers in the system.**

I. Introduction

Stirred by the remarkable popularity of Internet file-sharing software, distributed systems and networking research made peer-to-peer (P2P) systems a focal point of their recent studies. As opposed to P2P systems, conventional distributed systems typically consist of a *fixed* set of machines. During operation, occasionally (but rarely!) a small subset of machines might fail (crash or behave maliciously, depending on the model). Thanks to ingenious communication protocols these failures will be detected, and operable parts of the system will eventually be guided back to a save state.

In a P2P system, however, there is no fixed set of participating machines. Instead, a distributed P2P system is composed of a huge number of machines (peers) which join and leave the system at high rates. In P2P lingo, this high turnover of machines is called *churn*. For distributed systems with high churn the orthodox group communication schemes seem futile. In a P2P system with millions of peers where each participates in the system for a few hours on average, hundreds of membership changes take place every second. In such a system, it seems out of the question to achieve consensus which peers currently participate.

In spite of being a foremost difficulty in P2P systems, churn has not received the attention it deserves in the literature. With the exception of [12], P2P systems are instead analyzed against an adversary which can crash a functionally bounded number of random peers. Then, much in the esprit of self-stabilization or group communication, the P2P system is given sufficient time to recover.

In this paper we describe how to construct an efficient distributed hash table (DHT) which is resilient to churn. We assume that joins and leaves occur in a worst-case manner, i.e., an adversary chooses which peers to crash and how peers join. Thereby, the adversary does not need to wait until the system is recovered before it crashes the next group of peers. Instead, the adversary may crash peers continuously while the system is trying to stay alive. Our system remains fully functional in the presence of an adversary constantly attacking its weakest part. Such an adversary is countered by our system by continuously moving the remaining or newly joining peers towards the weakest areas.

Of course, we cannot allow our adversary to have unlimited capabilities. In particular, in any constant time interval, the adversary can at most add and/or remove $\Theta(\frac{\log n}{\log\log n})$ peers, n being the total number of peers presently in the system. Since the peer degree—or the *routing state* per peer—is also in $\mathrm{O}(\frac{\log n}{\log\log n})$, this is asymptotically optimal: If the adversary could remove as many peers as the the peer degree, it would be able to disconnect a peer completely from the system by crashing all the peer's neighbors. Our model covers an adversary which repeatedly takes down machines by a distributed denial of service attack, but only a bounded number of machines at each point in time. Our system is *synchronous* and we assume messages to be delivered timely, that is, in at most constant time between any pair of operational peers. It would be possible to adapt the system for an asynchronous environment; in this case, the propagation delay of the slowest message defines the notion of time which is needed for the adversarial model.

The basic structure of our P2P system is a pancake graph (cf Definition 1.1 and Figure 1) of order d. Each peer is part of a distinct pancake node; each pancake node consists of $\mathrm{O}(d^2)$ 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 the 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.

1-4244-0476-2/06/$20.00 ©2006 IEEE.

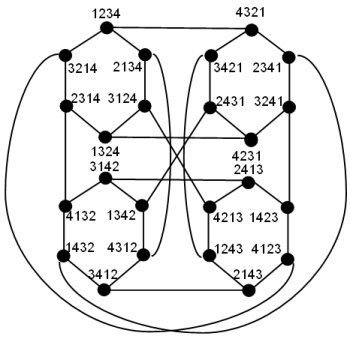

Fig. 1. A pancake graph of order 4 (P_4).

Definition 1.1: A *pancake graph* of order d is a graph $P_d = (V, E)$, with $V(P_d) = \{l_1 l_2 ... l_d \mid l_i \in \{1, ..., d\}, \forall i \neq j : l_i \neq l_j\}$, 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: $\rho_i(l_1 ... l_i ... l_d) := l_i l_{i-1} ... l_1 l_{i+1} ... l_d$. For $u, v \in V(P_d)$, it holds that $\{u, v\} \in E(P_d) \Leftrightarrow v = \rho_i(u)$ for $i \in \{2, ..., d\}$. P_d is a $(d-1)$-regular graph of diameter smaller than $2d$.

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

The balancing of peers among the pancake nodes can be seen as a dynamic token distribution problem [15] on the pancake. Each node of a graph having a certain number of tokens, the goal is to distribute the tokens along the edges of the graph such that all nodes end up with roughly the same number of tokens. While tokens are moved around, an adversary continuously adds and removes 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 adapt the order accordingly.

Apart from the pancake graph, our techniques can be applied to any of the popular P2P topologies. We decided to use the pancake graph for two reasons. First, we can show that the described method works for rather complex topologies such as the pancake graph. Second, it allows to obtain a system which minimizes the maximum of peer degree and network diameter. Based on the described structure, we get a fully scalable P2P system with peer degree and network diameter $\mathrm{O}(\log n / \log \log n)$, implying time complexity $\mathrm{O}(\log n / \log \log n)$ for the usual operations (e.g. search). At the same time, our system tolerates $\Theta(\log n / \log \log n)$ worst-case joins and/or crashes per constant time interval.

The road-map of the paper is as follows. After introducing our model (Section II) and discussing related work (Section III), the basic algorithms of our system together with their proofs are provided in Section IV. In Section V, we show how to compose these algorithms in order to build the complete P2P system. The paper is concluded in Section VI.

II. Model

We consider the *synchronous message passing model* where in each round, every peer can send a message to all its neighbors. Dynamic worst-case joins and leaves (churn) are modeled with an adversary $\mathcal{A}_{ADV}(J, L, \lambda)$ which may perform J arbitrary joins and L arbitrary leaves (crashes) in any time interval of length λ rounds. A joining peer π_1 is assumed to contact an arbitrary peer π_2 which already belongs to the system. In contrast to other networks where peers are bound to execute some finalizing protocols before leaving, we consider the rough model where peers depart or crash without notice.

III. Related Work

The pancake graph and the unsolved problem of computing its diameter was introduced in [7].[1] In terms of the group-theoretic model for network topologies introduced by Akers and Krishnamurthy [3], the pancake is an instance of a hierarchical *Cayley graph* [4]. It has the interesting property that there is no graph with both asymptotically smaller degree *and* smaller diameter; the degree can only be reduced further at the expense of a larger diameter, and vice versa. However, to the best of our knowledge, this is the first paper to address the issues of scalability, information aggregation, and token distribution on the pancake graph.

Over the last years, enough and to spare overlay networks with various interesting technical properties have been proposed (e.g. [2], [5], [6], [10], [11], [14], [16], [17], [20]). Because of the nature of P2P systems, fault-tolerance has been a prime issue from the beginning. The systems are usually robust against a large number of random faults. But after crashing a few peers, the systems are given time to recover again.

Resilience to worst-case failures has been studied by Fiat, Saia et al. in [8], [18]. They introduce a system where, with high probability, $(1-\varepsilon)$-fractions of peers and data survive the adversarial removal of up to half of all nodes. However, in contrast to our work the failure model is static. Moreover, the whole structure has to be rebuilt from scratch if the total number of peers changes by a constant factor.

Abraham et al. [1] address scalability and resilience to worst-case joins and leaves. 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 has some time to adapt.

The first paper treating arbitrarily concurrent worst-case joins and leaves is by Li et al. [13]. In contrast to our paper, 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. Leaving peers execute an appropriate "exit" protocol and do not leave before the system allows; crashes are not allowed.

[1] In no paper about pancake graphs a reference to [9] shall be missing: A paper that was originally written in 1976 by a Ph.D. student (later to become a famous scholar) and a 21 year old college dropout (later to become a famous entrepreneur).

To the best of our knowledge the only solution tolerating continuous joins and leaves is [12]. In [12] it is shown that a hyper-cubic topology can tolerate $\Theta(\log n)$ worst-case joins and/or crashes per constant time interval, where n is the total number of peers. In this paper—superficially—we improve the result of [12] by presenting a topology with better characteristics (faster search time and lower degree). However, we think our main contribution is to make a most intricate graph topology dynamic. Mastering the pancake, we believe, essentially gives a recipe for any recursively defined P2P topology, by simply applying our basic components (cf Section IV) as ingredients.

IV. Basic Components

A. Scaling

The order of the pancake graph is changed according to the total number of peers in the system. For the expansion, node $l_1...l_d \in V(P_d)$ splits into $d+1$ new nodes $\{(d+1)l_1l_2...l_d, l_1(d+1)l_2...l_d, ..., l_1l_2...l_d(d+1)\}$ 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 crucial 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.

1) Expansion: If the total number of peers in the system exceeds a certain threshold, each node $v = l_1...l_d \in V(P_d)$ splits into $d+1$ new nodes $\{v^{exp}_{(1)} := (d+1)l_1l_2...l_d, v^{exp}_{(2)} := l_1(d+1)l_2...l_d, ..., v^{exp}_{(d+1)} := l_1l_2...l_d(d+1)\}$ 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 by the knowledge about the neighbors of the original node $v \in V(P_d)$.

Lemma 4.1: 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 $i, j \in \{1, ..., d+1\}$, 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 $k \in \{2, ..., d+1\}$ 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: $u = \rho_k(v)$. ■

2) Reduction: If the total number of peers in the system falls below a certain threshold, all nodes $l_1...l_i(d+1)l_{i+1}...l_d \in V(P_{d+1})$ for $i \in [0, d]$ merge into a single node $l_1...l_d \in V(P_d)$. Unfortunately, we cannot reverse the expansion directly. Instead, the reduction works as follows. First, the following *dominating set* on P_{d+1} is computed: Every node $v = l_1...l_{d+1}$ having $l_1 = d+1$ becomes a dominator. We will call a dominator plus its adjacent (dominated) nodes a *cluster*. In the following, let $v^{dom}_{(1)} = (d+1)l_1...l_d$ be a dominator and $v^{dom}_{(i)} = \rho_i(v^{dom}_{(1)}) = l_{i-1}l_{i-2}...(d+1)l_i...l_d$ its neighbor with prefix-inversion of length i, for $i \in [1, d+1]$. The idea is to contract each cluster with dominator $v^{dom}_{(1)} = (d+1)l_1...l_d$ to a single node $v = l_1...l_d \in V(P_d)$. 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 4.2: 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 $v = l_1l_2...l_{d+1}$. Assume that $l_i = d+1$ for some $i \in \{1, ..., d+1\}$. 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 4.2, 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 $v^{dom}_{(1)} = (d+1)l_1...l_d$ sends its dominated node $v^{dom}_{(i+1)}$ to the cluster with dominator $(d+1)\rho_i(l_1...l_d)$, for $i \in [2, d]$.

It holds that after the exchange of the dominated nodes, (i) each cluster with dominator $v^{dom}_{(1)} = v^{exp}_{(1)} = (d+1)l_1...l_d$ which will contract to node $v = l_1...l_d$ consists of the nodes $v^{exp}_{(1)} = (d+1)l_1...l_d, v^{exp}_{(2)} = l_1(d+1)...l_d, ..., v^{exp}_{(d+1)} = l_1...l_d(d+1)$, 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 $\rho_{i-1}(v^{dom}_{(i)}) = \rho_{i-1}(l_{i-1}...l_1(d+1)l_i...l_d) = v^{exp}_{(i)}$, and that before the transfer, $v^{exp}_{(i)}$ belonged to the cluster dominated by $\rho_i(v^{exp}_{(i)}) = (d+1)l_{i-1}...l_1l_i...l_d$ which will reduce to node $\rho_{i-1}(v)$. Thus, after the exchange, the following lemma holds.

Lemma 4.3: 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.

B. Information Aggregation

As stated, each pancake node is simulated by several peers, and the pancake's order is adapted according to the total number of peers in the system. In this section, we present an algorithm $\mathcal{A}_{IA}$ which allows to count the total number of peers in the pancake's nodes. In our description, we use the term *token* rather than peer.

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 4.4: $\mathcal{A}_{IA}$ 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 $v = l_1 ... l_d$ 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 allows to compute 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 $l_{i+2}l_{i+3}...l_d$ with v. Performing a ρ_{i+2} prefix-inversion yields a member for each sub-graph with postfix $l_{i+3}l_{i+4}...l_d$ of length $d-(i+2)$. Therefore, combining the information of the sub-graphs yields the total number of tokens in $P_{i+2}(v)$. ■

In our system, $\mathcal{A}_{IA}$ 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.

C. Token Distribution

Ideally, the number of peers per pancake node should be roughly equal for all nodes. Because peers join and leave, it is necessary to constantly adapt the assignment of peers to nodes. The problem of assigning peers to nodes is closely related to the *token distribution problem* as introduced in [15]. Given a graph G and a number of tokens at each node of G, the goal is to find a distributed algorithm which moves tokens along the edges of G such that in the end, the tokens are distributed equally among all nodes of G.

The problem is of prime importance in the field 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 [19].

In the context of this paper, we look at a dynamic token distribution problem on the pancake graph where in each step, tokens (peers) can be added and removed at arbitrary nodes. The objective is to constantly move tokens along edges such that at all times, all pancake nodes have roughly the same number of tokens.

Formally, the goal is to minimize the maximum difference of the number of tokens of any two pancake nodes, denoted by the *discrepancy* ϕ. Analogously to the information aggregation algorithm of Section IV-B, 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 as described in Section IV-A 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 entries $i+1$ to d are fixed for all nodes of $P_i(v)$ by definition.

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 as in Section IV-B. 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 4.5: $\mathcal{A}_{TD}$ 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 i^{th} entry. Let l_i be the i^{th} 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 4.6: 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, we can look at each token independently. By Lemma 4.5, 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. ■

Up to now, 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 thus have to extend the analysis to integer tokens. We assume that in line 4, tokens are distributed as good 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 4.7: 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 4.5, 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 does solely depend 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. ■

Combining Lemmas 4.5, 4.6, and 4.7, we obtain the following theorem about the dynamic integer token distribution algorithm.

Theorem 4.8: The discrepancy ϕ of the dynamic integer token distribution algorithm is at most $\phi \leq 4d + 3(J+L)$.

We conclude this section with a few considerations about an actual implementation of $\mathcal{A}_{TD}$. The algorithm 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.

D. Node Representation

The algorithms provided so far have all been described on the level of pancake graphs. In this section, we take a more

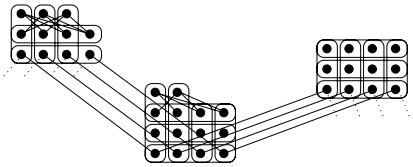

Fig. 2. The peers of a pancake node are arranged as a grid with $d+1$ columns. A peer has connections to all peers in its row plus to all peers in its column. The pancake's edges are represented by a matching between the peers of the bottom row.

detailed look at the internals of our system. As stated, our system *simulates* the pancake topology and a pancake node consists of several peers. But how are peers of the same node connected to each other (*intra-connections*)? And how are peers connected to peers in adjacent nodes (*inter-connections*)?

In Section IV-D.1, we present the representation of the pancake's nodes and edges. Section IV-D.2 then gives an algorithm which allows to maintain these structures against a concurrent adversary. Finally, we give the algorithms for expansion and reduction of the pancake's order (Sections IV-D.3 and IV-D.4). Note that—due to space constraints—we omit the peer-level description of some components, for example the token distribution or also the information aggregation algorithm. However, these operations are straight-forward and can be done with similar techniques.

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 Figure 2.

2) Grid Maintenance: In this section, we describe how to maintain the grid against concurrent adversarial churn. Our algorithm $\mathcal{A}_{GRID}$ needs several rounds. The idea is as follows: 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 may have crashed by the concurrent adversary in the meantime. That is, by using enough 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 may be removed completely and that there is always a complete column in the presence of a concurrent adversary $\mathcal{A}_{ADV}(\frac{d}{2}, \frac{d}{2}, 5)$ which may add and remove at most $\frac{d}{2}$ peers in any time period of 5 rounds. Moreover, also the pancake's edges may be repaired in constant time since we ensure that two adjacent pancake nodes always have at least two living adjacent core peers which may 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 necessary information 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 according to the snapshot's state.

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 (cf Section IV-A). 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 Section IV-A, 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 definitely 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 established, 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 $v = l_1...l_d$ with grid G_v. The column $G_v[i, \cdot]$ for $i \in [1, d+1]$ will form the new node $v_{(i)}^{exp} = l_1...l_{i-1}(d+1)l_i...l_d$. Since peers of the i^{th} column $G_v[i, \cdot]$ are completely connected, the expansion can be performed in two rounds: It is straightforward 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 i^{th} 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) 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 Section IV-A).

Similarly to the notation introduced in Section IV-A, 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 Section IV-A).

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 Section IV-A). 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.

V. 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 $hash(ID)$. However, a data item is not copied to *all* peers in that node, but only replicated on the core at the bottom row. 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. Of course, this solution implies a certain load imbalance in the sense that data is not evenly distributed among the peers. However, we will not discuss how to remedy this, as our focus here is mainly on the provable fault-tolerance of our system. Finally, observe that routing is simple in the pancake system: Assume that a peer in a node $u = l_1 l_2...l_d$ wants to find a data item which hashes to a node $v = \widehat{l_1}\widehat{l_2}...\widehat{l_d}$. The lookup operation proceeds by correcting one "coordinate" at a time, starting at the back: From node $u = l_1 l_2...l_d$ the request is forwarded to node $l_d...l_{j+1}l_1l_2...l_{j-1}\widehat{l_d}$, etc.

We now describe how the components introduced in Section IV are assembled to form a P2P system resilient to an adversary $\mathcal{A}_{ADV}(\Theta(\frac{\log n}{\log\log n}), \Theta(\frac{\log n}{\log\log n}), 1)$. Our system permanently runs $\mathcal{A}_{IA}$ to estimate the total number of peers in the system (cf Section IV-B) and adapts the pancake's order accordingly (cf Section IV-A), $\mathcal{A}_{TD}$ to distribute the peers evenly among the pancake's nodes (cf Section IV-C), and $\mathcal{A}_{GRID}$ to maintain the grid (cf Section IV-D). 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 enough 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 4.8, the difference between the total number of peers at any two nodes is bounded by $\mathrm{O}(d)$ if there are $\mathrm{O}(d)$ joins and leaves per time unit, the following theorem holds.

Theorem 5.1: Our pancake P2P system guarantees peer degree and network diameter $\mathrm{O}(d)$ in the presence of an adversary which 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 $d = \Theta(\frac{\log n}{\log \log n})$, where n is the total number of peers in the system.

VI. Conclusions

We have described how to construct a P2P system which maintains desirable properties such as low peer degree and low network diameter against a powerful, concurrent adversary which has complete visibility of the entire state of the system. It has been shown that the fault tolerance is asymptotically optimal as the robustness of any topology is upper bounded by its peer degree.

From a practical point of view, our system could be optimized in various respects. For instance the total number of messages sent can be made adaptive to the dynamics of the system: As long as only few joins and leaves happen, the message complexity can be kept low by communicating the information about the changes only when really needed. Further, the load imbalance due to the division of peers into core and peripheral peers could be alleviated using a more sophisticated system. However, since our focus is on proving dynamic worst-case fault-tolerance rather than on developing a ready-to-use system, we did not consider these practical issues.

While the various new algorithms for the pancake graph may be of independent interest, our main contribution is the introduction of techniques which allow to make a most intricate topology dynamic. Having mastered the pancake graph, we believe that applying our basic components as ingredients gives a recipe for any P2P topology.

Acknowledgments

Research supported by the Swiss National Science Foundation and the Hasler Stiftung.

References

[1] I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, and E. Pavlov. A Generic Scheme for Building Overlay Networks in Adversarial Scenarios. In *Proc. 17th Int. Symp. on Parallel and Distributed Processing (IPDPS)*, 2003.

[2] I. Abraham, D. Malkhi, and O. Dobzinski. LAND: Stretch (1 + ε) Locality-Aware Networks for DHTs. In *Proc. 15th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 550–559, 2004.

[3] S. B. Akers and B. Krishnamurthy. A Group-Theoretic Model for Symmetric Interconnection Networks. *IEEE Transactions on Computing*, 38(4):555–566, 1989.

[4] F. Annexstein, M. Baumslag, and A. L. Rosenberg. Group Action Graphs and Parallel Architectures. *SIAM J. Comput.*, 19(3):544–569, 1990.

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

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

[7] H. Dweighter (a.k.a. J. E. Goodman). *American Mathematical Monthly*, 82, 1975.

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

[9] W. Gates and C. Papadimitriou. Bounds for Sorting by Prefix Reversal. *Discrete Math.*, 27:47–57, 1979.

[10] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *Proc. 4th USENIX Symp. on Internet Technologies and Systems (USITS)*, 2003.

[11] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proc. of ACM ASPLOS*, November 2000.

[12] F. Kuhn, S. Schmid, and R. Wattenhofer. A Self-Repairing Peer-to-Peer System Resilient to Dynamic Adversarial Churn. In *Proc. 4th Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.

[13] X. Li, J. Misra, and C. G. Plaxton. Active and Concurrent Topology Maintenance. In *Proc. 18th Ann. Conference on Distributed Computing (DISC)*, 2004.

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

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

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

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proc. of ACM SIGCOMM 2001*, 2001.

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

[19] B. A. Shirazi, K. M. Kavi, and A. R. Hurson. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Science Press, 1995.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM Conference*, 2001.