ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Diploma Thesis

A Churn-Resistant P2P-System Based on the Pancake Graph

Joest M. Smit

Prof. Dr. Roger Wattenhofer
Fabian Kuhn
Stefan Schmid

Distributed Computing Group
ETH Zurich, Switzerland

November 2004 - March 2005

Abstract

Since many years, peer-to-peer systems are very popular and the number of their users is still increasing. Also in the research of computer science, this topic has achieved much attention. The analysis of fault tolerant P2P-systems mostly covers random joins and leaves in a static model, i.e. a P2P system tolerates a number of simultaneous random faults. However, in real P2P system, the set of peers connected to the system, continuously changes. For this, a dynamic model seem to be more appropriate.

In this diploma thesis, a P2P system is presented, that can defend a malicious adversary, joining and leaving peers in the system. In contrast to other models, the adversary can do this continuously in a worst case manner, while the system tries to stay fully functional.

Contents

1 Introduction **5**
- 1.1 Model 7
- 1.2 Related Work 7

2 Basic Components **9**
- 2.1 Scaling 9
  - 2.1.1 Expansion 9
  - 2.1.2 Reduction 9
- 2.2 Information Aggregation 10
- 2.3 Token Distribution 11
- 2.4 Node Representation 14
  - 2.4.1 The Grid 14
  - 2.4.2 Edges 15

3 Algorithms **17**
- 3.1 Introduction 17
- 3.2 Grid Maintenance 18
- 3.3 Updating Inter-Node Connections 20
- 3.4 Expansion 22
  - 3.4.1 Matching Establishment 22
  - 3.4.2 Matching maintenance 25
  - 3.4.3 Expand Step 26
- 3.5 Reduction 27
- 3.6 Token Distribution on the Grid Level 33
- 3.7 Information Aggregation Algorithm in Our System 37

4 The System **41**
- 4.1 Overview 41
- 4.2 Thresholds 43
- 4.3 Properties of the Pancake System 46
- 4.4 Data and Routing 47

5 Conclusions **51**

A Acknowledgements **55**

Chapter 1

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 will be no fixed set of participating machines. Instead, a distributed P2P system is composed of a huge number of machines (peers) who 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 peers join and leave the system 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 [13], P2P systems are instead analyzed against an adversary who 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 an efficient P2P system which is resilient to churn. We assume that joins and leaves occur in a worst-case manner. In particular, an adversary can remove and add a bounded number of peers. The adversary chooses which peers to crash and how peers join. However, we assume that a joining peer knows a peer which already belongs to the system. Moreover, 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 such an adversary which constantly attacks its weakest part. For example, an adversary could insert a crawler into the P2P system, learn the topology, and then repeatedly crash selected peers, attempting to partition the P2P network. Such an adversary is countered by our system by continuously moving the remaining or newly joining peers towards the weakest areas.

Of course, we can not allow our adversary to have unlimited capabilities. In particular, in any constant time interval, the adversary can at most add and/or remove

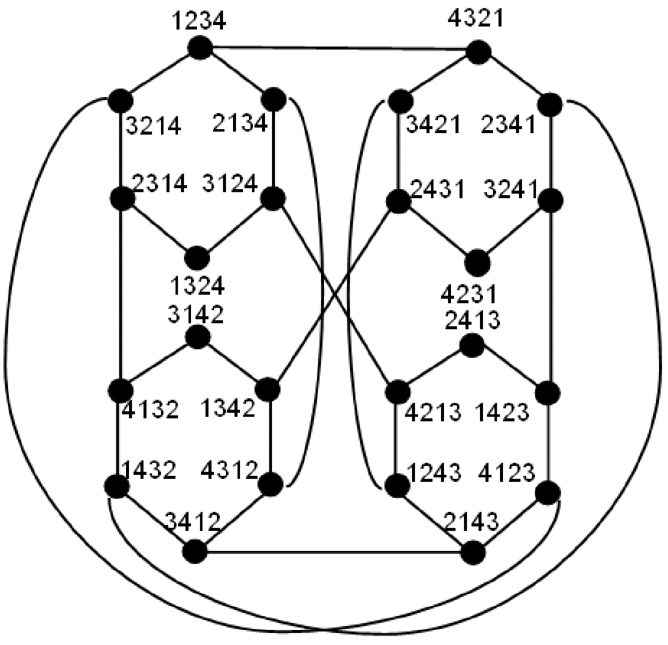

Figure 1.1: A pancake graph of order 4 (P_4).

$O(\frac{\log n}{\log\log n})$ peers, n being the total number of peers presently in the system. This 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, i.e., in at most constant time between any pair of operational peers. Note however that if nodes are synchronized locally, our algorithm also runs in an asynchronous environment. Thereby, 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 1.1). Each peer is part of a distinct pancake node; each pancake node consists of $O((\frac{\log n}{\log\log n})^2)$ 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 and to peers of the neighboring pancake nodes. 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.

Definition 1.1. *A* Pancake Graph *of order d is a graph $P_d = (V, E)$, with $V(P_d) = \{l_1 l_2 ... l_d \,|l_i \in \{1, ..., d\}, \forall i \neq j : l_i \neq l_j\}$, i.e., $V(P_d)$ is the set of all permutations on d elements. Let ρ_i denote a prefix-inversion of length i, i.e. $\rho_i(l_1...l_i...l_d) := l_i l_{i-1}...l_1 l_{i+1}...l_d$. We have $\{u, v\} \in E(P_d) \Leftrightarrow v = \rho_i(u)$.*

The balancing of peers among the pancake nodes can be seen as a dynamic token distribution problem [16] on the pancake. Each node of a graph has a certain number of tokens, the goal is to distribute the tokens along the edges of the graph such that all nodes end up with roughly the same number of tokens. While tokens are moved around, an adversary continuously 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 accordingly adapt the order.

Based on the described structure, we get a fully scalable, efficient P2P system which tolerates $\mathrm{O}(\frac{\log n}{\log\log n})$ worst-case joins and/or crashes per constant time interval. Moreover, peers have $\mathrm{O}(\frac{\log n}{\log\log n})$ neighbors, and the usual operations (e.g. search) take time $\mathrm{O}(\frac{\log n}{\log\log n})$.

1.1 Model

The *synchronous message passing model* is considered where in each round, every peer can send a message to all its neighbors. The ongoing churn is modelled with an adversary $\mathcal{A}_{ADV}(J, L)$ which may perform J arbitrary joins and L arbitrary leaves (crashes) per time interval of unit length. A joining peer π_1 is assumed to contact an arbitrary peer π_2 which already belongs to the system. In contrast to other systems where peers have to do some finalizing operations before leaving, we consider the more general case where peers depart or crash without notice.

1.2 Related Work

The pancake graph and the famous unsolved problem of computing its diameter has been introduced by [6], and has been analyzed in several papers [5, 8, 10, 11]. However, to the best of our knowledge, this is the first paper that addresses 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, 3, 4, 9, 12, 15, 17, 18, 21, 22]). 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. Churn has been addressed in [19] from an experimental point of view.

Resilience to worst-case failures has been studied by Fiat, Saia et al. in [7, 20]. They introduce a system where, with high probability, $(1-\varepsilon)$-fractions of peers and data survive the adversarial deletion 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. [14]. 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 this; crashes are not allowed.

To the best of our knowledge the only paper which tolerates continuous joins and leaves is [13]. In [13] it is shown that a hypercubic topology can tolerate

$O(\log n)$ worst-case joins and/or crashes per constant time interval. In this paper—superficially—we improve the result of [13] 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. Majoring the pancake, we believe, essentially gives a recipe for any P2P topology, by simply applying our basic components (see Chapter 2) as ingredients.

Chapter 2

Basic Components

2.1 Scaling

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

To be useful for our application, the order change of the pancake has to fulfill a crucial requirement: A node must be able to compute its new neighbors *locally*, i.e., based on the information about the neighbors in the graph before the order changed. We will now describe the expansion and the reduction in detail and show that this criterion is indeed fulfilled in both cases.

2.1.1 Expansion

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

Lemma 2.1. *Consider two arbitrary nodes u and v. It holds that if $\{u_{(i)}^{exp}, v_{(j)}^{exp}\} \in E(P_{d+1})$ for some $i, j \in \{1, ..., d+1\}$, then $\{u, v\} \in E(P_d)$ or $u = v$.*

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

2.1.2 Reduction

If the total number of peers per node on average falls beyond a certain threshold, all nodes $l_1 ... l_i(d+1)l_{i+1} ... l_d \in P_{d+1}$ for $i \in [0, d]$ merge into a single node $l_1 ... l_d \in P_d$. This 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 $v_{(1)}^{dom} = (d+1)l_1...l_d$ be a dominator and $v_{(i)}^{dom} = \rho_i(v_{(1)}^{dom}) = l_{i-1}l_{i-2}...(d+1)l_i...l_d$ its neighbor with prefix-inversion of length i. The idea will be to contract each cluster with dominator $v_{(1)}^{dom} = (d+1)l_1...l_d$ to a single node $v = l_1...l_d \in P_d$. However, 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 never adjacent.

Lemma 2.2. *Consider the graph P_{d+1}. The $d!$ nodes of P_{d+1} with first number $l_1 = d+1$ 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. Clearly, two nodes having $l_1 = d+1$ cannot be adjacent. If $i \neq 1$, there is exactly one neighbor of v which is a dominator, namely node $u = \rho_i(v)$. $\square$

According to Lemma 2.2, each node belongs to exactly one cluster, hence the contraction operation is well-defined.

However, as stated, we additionally need to exchange dominated nodes between adjacent clusters. This works as follows: The cluster with dominator $v_{(1)}^{dom} = (d+1)l_1...l_d$ sends its dominated node $v_{(i)}^{dom}$ to the cluster with dominator $(d+1)\rho_i(l_1...l_d)$, for $i \in [2, d]$.

We will now show that after the exchange of the dominated nodes, (1) each cluster with dominator $v_{(1)}^{dom} = v_{(1)}^{exp} = (d+1)l_1...l_d$ which will reduce to node $v = l_1...l_d$ consists of the nodes $v_{(1)}^{exp} = (d+1)l_1...l_d, v_{(2)}^{exp} = l_1(d+1)...l_d, ..., v_{(d+1)}^{exp} = l_1...l_d(d+1)$, and (2) the dominated node $v_{(i)}^{exp}$ for $i \in [3, d+1]$—before being transferred to the cluster dominated by $v_{(1)}^{dom}$—belonged to the cluster that will form the new node $\rho_i(v)$. To see this, note that node $v_{(i)}^{dom}$ is replaced by $\rho_{i-1}(v_{(i)}^{dom}) = \rho_{i-1}(l_{i-1}...l_1(d+1)l_i...l_d) = v_{(i)}^{exp}$, and that before the transfer, $v_{(i)}^{exp}$ belonged to the cluster dominated by $\rho_i(v_{(i)}^{exp}) = (d+1)l_{i-1}...l_1l_i...l_d$ which will reduce to node $\rho_{i-1}(v)$. Thus, after the exchange, the cluster which will contract to node v consists of the nodes v would also expand to, and a cluster has information about each of its future neighbors.

2.2 Information Aggregation

The order of our pancake is adapted according to the total number of peers in the system. In this chapter, we present an algorithm $\mathcal{A}_{IA}$ which allows to count the total number of tokens (peers) at the pancake's nodes. Let $P_i(v)$ denote the sub-graph of the pancake graph P_d consisting of those nodes that share a postfix of length $d-i$ with a given node v. (Note that the graph induced by $P_i(v)$ is also a pancake graph, namely 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: $|P_{i+1}(v)| = |P_i(v)| + \sum_{j=1}^{i} |P_i(\rho_{i+1}(\rho_j(v)))|$, where $|\cdot|$ 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 2.3. *$\mathcal{A}_{IA}$ 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 trivially holds.

$i \rightarrow i+1$: 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 $P_{i+2}(v)$.

To see this, note that the nodes $\rho_j(v)$ for $j < i+2$ all have a different number at the first place and share the postfix $l_{i+2}l_{i+3}...l_d$ with v. Performing a ρ_{i+2} prefix-switch 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. Even it could be run pipelined, i.e. all phases concurrently and thus supplying a result after each phase, this will not be done, since the number of integer that would be received by a peer would be in $\Theta(d^2)$. The latest result, before the arrival of the next result, is thus delayed by at most $2d - 3$ phases.

2.3 Token Distribution

As stated, each pancake node is simulated by a number of peers. 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. To problem of assigning peers to nodes is closely related to the token distribution problem as introduced in [16]. Given a graph G and a number of tokens at each nodes 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. 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 inserted and deleted 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 Chapter 2.2, our token distribution algorithm 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 is given in Algorithm 1. We assume that we have given a dominating set as described in Chapter 2.1 for each pancake $P_i(v)$. The dominators could e.g. be all nodes of $P_i(v)$ which have the largest of the first i coordinates at the first position. Note that coordinates $i + 1$ to d are fixed for all nodes of $P_i(v)$ by definition.

Algorithm 1 Pancake Token Distribution (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 Chapter 2.2. After the i^{th} iteration of Algorithm 1, 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 Algorithm 1, 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 inserted or deleted 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 inserted or deleted during the algorithm (static token distribution). We will then analyze the effects of adversarial insertions and deletions and of integer tokens.

Lemma 2.4. *Algorithm 1 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 u, all nodes of $P_{i-1}(u)$ have the same number of tokens $t_{i-1}(u)$. The pancakes $P_{i-1}(u)$ of order $i - 1$ belonging to $P_i(v)$ can be characterized by their i^{th} coordinate. Let l_i be the i^{th} coordinate of the nodes of $P_{i-1}(u)$. In line 2 of Algorithm 1, a node u of $P_{i-1}(u)$ modes all tokens to $\rho_i(u)$, that is, all tokens are moved to a node with l_i as its first coordinate. Hence, after line 2, all nodes of $P_i(u)$ with first coordinate l_i have $t_{i-1}(u)$ 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 coordinate 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 Algorithm 1. 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 2.5. *If in every iteration of Algorithm 1 at most J tokens are inserted and at most L tokens are deleted, 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 2.4, each token which is inserted before the first iteration of the algorithm is distributed equally among $1/i! \leq 2^i$ nodes after iteration i. A token which is inserted after iteration j is distributed among $i!/j! \leq 2^{i-j}$ nodes after iteration i. All tokens which were inserted before the last complete execution of Algorithm 1 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 Algorithm 1 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. Because 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 number of tokens between the heaviest and the lightest node becomes $3J$. For deleted tokens the same argumentation as for inserted tokens holds. $\square$

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, 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 2.6. *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 Algorithm 1. 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 nodes, this 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 2.4, fractional tokens are distributed equally among all nodes after their first complete execution of Algorithm 1, that is, after less than $2d$ iterations. Therefore, the number of fractional tokens at each node does only 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. $\square$

Combining Lemmas 2.4, 2.5, and 2.6, we obtain the following theorem about the dynamic integer token distribution algorithm.

Theorem 2.7. *The discrepancy ϕ of the dynamic integer token distribution algorithm is at most $\phi \leq 4d + 3(J + L)$.*

We end this chapter with a few considerations about an actual implementation of Algorithm 1. The algorithm is formulated in the form which makes the proofs of this chapter 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 Algorithm 1 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.

2.4 Node Representation

Our system *simulates* the pancake topology and a pancake node consists of several peers. In this chapter we will first present the internal structure of a node (Chapter 2.4.1) plus the representation of the pancake's edges (Chapter 2.4.2).

2.4.1 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 $|v|$ be the total number of peers in node v and let $R := \lfloor |v|/(d + 1) \rfloor$. 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 $|v|$ mod $(d+1)$ peers—from now on called *extra peers*—are located in an incomplete additional row $G_v[i, R]$ for $i \in [0, |v|$ mod $(d+1)]$. Inside a node, the peers are connected as follows (*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: Assume the grid having two rows and one extra-peer, the extra-peer would only be connected to two peers. In order to avoid this weakness, the extra peers are also assumed to be full members of the highest complete row $R - 1$, i.e., we also have connections between $G_v[i, R]$ for $i \in [0, |v|$ mod $(d + 1)]$ and *all* peers $G_v[j, R - 1]$ for $j \in [0, |v|$ mod $(d + 1)]$. When we state that a peer sends a message to all of it's row members, we thus mean for the row $R - 1$, that the senders send the message also to the extra-peers and, vice versa, also the extra-peers send the corresponding message to the members of row $R - 1$.

Definition 2.1. *A grid as described above and containing at least $2d+2$ peers is called a* fully repaired *grid.*

2.4.2 Edges

Having described the pancake's nodes, we now 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., $\{u, v\} \in P_d$, 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 zero) the *core* of the corresponding node. Thus, we can say that two nodes are connected by a *matching between their cores*.

Definition 2.2. *We call the matching of a node in P_d* fully repaired, *when each column in the grid has a core-peer, and all core peers have a connection to their matching peers in all neighbored nodes.*

Definition 2.3. *The pancake system is called* fully repaired, *when all grids as well as their inter-node connections are fully repaired, and the discrepancy between any two nodes is bounded by $4d + 3(J + L)$.*

Lemma 2.8. *Assumed the number of peers in any grid of the pancake system is bounded by $\mathrm{O}(d^2)$, the following statement holds: For any peer in a fully repaired pancake system, the out-degree is bounded by $\mathrm{O}(d)$.*

Proof. A peer in the pancake system is connected to its row neighbors, which are at most $2d$ (in the top row, when there are d extra peers) and its column neighbors. For the row neighbors there is nothing to prove: Each node has just d. Since the number of peers per node is assumed to be bounded by $\mathrm{O}(d^2)$, the number of rows in the grid, and thus the number of column neighbors is bounded by $\mathrm{O}(d)$, since we have $d + 1$ columns. Finally, the core peers are connected to one core peer of each neighbored node. Since each node in P_d has exactly $d - 1$ neighbors, the statement holds. $\square$

Chapter 3

Algorithms

3.1 Introduction

In this chapter the algorithms that are needed for the pancake system are presented in detail. First the algorithm $\mathcal{A}_{GRID}$ is presented (Chapter 3.2), which will be used to repair the grid, followed by $\mathcal{A}_{EDGE}$ (Chapter 3.3), which updates the inter-node connections. Then the algorithms that are needed for the expansion step, the reduction step (Chapter 3.4), the token distribution (Chapter 3.6) and finally the information aggregation (Chapter 3.7) follow.

For understanding this chapter, it is crucial to have an idea of how the system finally will work and how the adversary will be modelled. For this, we here present the idea of the *phase*. A *phase* is the time interval in which various algorithms are run, depending of the state of the system. The first algorithms of a *phase* are always the same. Starting with $\mathcal{A}_{GRID}$ and $\mathcal{A}_{EDGE}$, in the following the algorithms for the token distribution $\mathcal{A}_{TD}$, and the information aggregation($\mathcal{A}_{IA}$) are run. Finally, depending on the current result of the token distribution algorithm, the expand step may be prepared (c.f. Chapter 3.4) or executed, or the reduction algorithm may be run. Then a new phase begins with $\mathcal{A}_{GRID}$. We will restrict our adversary to $d/2$ joins and leaves during any time interval that corresponds to the longest possible phase (c.f. Chapter 4). We will prove that $\mathcal{A}_{GRID}$ and $\mathcal{A}_{EDGE}$ will fully repair the pancake system, with respect to the joins and leaves of the last phase. Thus for all following algorithms, the pancake system will be fully repaired up to $d/2$ joins and leaves, which may occur at arbitrary time during the current phase. For this, correctness will be proved for at most $d/2$ joins and leaves during the algorithm run. Note that this is sufficient, even if some of the joins and leaves occur before the run of the algorithm: The information of the possible joins and leaves before the start of these algorithms is simply not used. Other algorithms running during the same phase can be neglected. Thus for proving properties of any algorithm $\mathcal{A}_X$, we think of the the three algorithms $\mathcal{A}_{GRID} \rightarrow \mathcal{A}_{EDGE} \rightarrow \mathcal{A}_X$ continuously running on each grid in P_d. This is possible, as long as it can be guaranteed, that all algorithms $\mathcal{A}_X$ that run during a phase, will end with a pancake system that is fully repaired up to the churn during it's run.

In the following, unless stated otherwise, with 'all peer in a grid/row/column', we mean all peers that were part of the grid on the beginning of the phase. These nodes are known by the corresponding (neighbored) peers in the grid, based on a *snapshot* in the beginning of the grid maintenance algorithm.

3.2 Grid Maintenance

In this chapter, we describe how to maintain the grid against continuous adversarial churn. Our algorithm $\mathcal{A}_{GRID}$ takes 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 only 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 take the crashed and newly joined peers only into account when the maintenance algorithm begins again with the first round. We thus accept that a newly joined peer will be disconnected, when the peer to which it joins leaves before a new run of $\mathcal{A}_{GRID}$ is started

$\mathcal{A}_{GRID}$ consists of two parts. In the first part, the following information is broadcasted 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 rows $R-1$ and $R-2$. 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 left peers (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, decrementing the number of rows ($R := R-1$). 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, perhaps 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 necessary neighbor information. With an adversary which may remove and insert at most $d/2$ peers during any time period of 4 rounds, we can guarantee that no row may be deleted completely and that there is always a complete column. Note that we refer to the joins and leaves as to the joins and leaves at the start of $\mathcal{A}_{GRID}$. As mentioned above, the churn that happens during the run of $\mathcal{A}_{GRID}$ is ignored.

We now give the detailed description of $\mathcal{A}_{GRID}$. Thereby, the following notation is used: We will 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.

ROUND 1

Outline: Starting to broadcast the positions where peers have left, the IP addresses of the joining peers, and the IP addresses of the extra peers plus row $G_v[\cdot, R-1]$ and $G_v[\cdot, R-2]$.

Sent Messages: 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

Outline: The broadcast is continued along the columns.

Sent Messages: Each peer at position $G_v[x, y]$, $y < R-2$ sends the addresses of the joiners in its row plus the information in which columns peers in its row have left to $G_v[x, \cdot]$. Each peer at position $G_v[x, y]$, $y \geq R-2$ sends all IPs of it's row, together with the corresponding grid coordinates to $G_v[x, \cdot]$. The joiners are sent without grid coordinates.

ROUND 3

Outline: The broadcast along the rows completes the information dissemination.

Sent Messages: Each peer at position $G_v[x, y]$ forwards the information received in ROUND 2 to the peers $G_v[\cdot, y]$, including the joining peers that joined the row during the last phase.

ROUND 4

Outline: The new form of G_v is computed locally.

Local Computation: Each peer at $G_v[x, y]$ computes the new positions of the joiners, extra-peers and the top-row. If the number of joiners and extra-peers exceeds the number of leaves by more then $d + 1$, the number of rows in the grid is increased. If on the other hand the number of gaps exceeds the number of joiners and extra-peers, the row number is decreased. The computation is not specified here. We just assume that the computation is done unambiguous and similar in all nodes.

Sent Messages: 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.

The following lemma is crucial for all later described algorithms: When $\mathcal{A}_{GRID}$ runs continuously on the grid and an adversary may joins or leaves at most $d/2$ peers during any time interval that corresponds to the length of one run of $\mathcal{A}_{GRID}$, the grid can be maintained despite the adversarial churn: The grid is thus always repaired up to the joins and leaves during the current and the last run of $\mathcal{A}_{GRID}$ as long as there are enough peers in the grid. Since other algorithms may be run in the same phase, a following algorithm can rely on a grid that is repaired up to $d/2$ joins and leaves.

Lemma 3.1. *Let the algorithm $\mathcal{A}_{GRID}$ run continuously on an initially fully repaired grid. Let further the number of joins and leaves in any time interval of 4 rounds be bounded by $d/2$, and the number of peers in the system always be at least $2d + 2$. Then the following statement holds: The system is always fully repaired up to the joins and leaves since the beginning of the last run of $\mathcal{A}_{GRID}$.*

Proof. We first show that the presented algorithm can repair a grid that is fully repaired up to d joins and leaves in quiet rounds, that is, no further joins and leaves may occur during the run of the algorithm. First we observe, that the maximum of d leaves and the minimum of $2d + 2$ peers in the grid guarantee at least one complete column during each complete run of $\mathcal{A}_{GRID}$, let it be column c, and thus a grid-diameter of at most 3. In ROUND 1, each peer at position $G_v[c, \cdot]$ receives all IP'addresses from its row, additionally, the peer at $G_v[c, R - 1]$ gets all ID's of row $G_v[\cdot, R]$, if there are extra peers in the grid. As the peers can compute from the received IP-addresses, which row-members are missing, also the grid coordinates of left row members are known. At least the peers at position $G_v[\cdot . c]$ then can then send the complete information as described in ROUND 2 to their column neighbors, and in ROUND 3, all peers of column

c are informed about all the information that has to be broadcasted through the grid. After sending this information into the rows in ROUND 3, it is guaranteed that all peers in the grid receive in ROUND 4 the necessary information for the computation: their row and column neighbors, the IPs of the extra-peers, and the complete information about row $R-1$ and $R-2$. The computation is not presented in detail. But since an order on the peer IP-addresses as well as the grid positions can be defined, it can obviously be done unambiguous. To conclude the first part of the proof, it has to be shown, that all peers indeed receive their new row- and column neighbors. Each peer in the grid can compute the following: It's new grid position, all peers in the rows with row number at least $R-2$ including the corresponding grid coordinates. Finally, all IP-addresses and coordinates from peers that will change their position. All peers that change their column receive the missing column information from a peer in the column. For each peer, being newly on a grid position with row number at most $R-3$, gets its row information from the peers in that row.

Let now run the grid maintenance algorithm repeatedly on an initially fully repaired grid. The number of joins and leaves on the grid are restricted to $d/2$ in any time interval of 4 rounds and finally the number of peers in the grid is always at least $2d+2$. After the first run, the number of joins and leaves is at most $d/2$. These can be repaired during the second run of $\mathcal{A}_{GRID}$, since the total number of leaves are restricted to d during both runs of $\mathcal{A}_{GRID}$. Note that it is not necessary to know in advance which column will be complete during the whole run. We conclude, that in the second run, all joins and leaves that occurred during the first run are repaired and thus the situation after the second run of $\mathcal{A}_{GRID}$ is the same as after the first run. Of course, this is also true for all following runs of $\mathcal{A}_{GRID}$. □

Since in our pancake-system, we bounded the out-degree by $\mathrm{O}(d)$, also the total number of different information a peer receives should be bounded by $\mathrm{O}(d)$. This is stated in the following lemma.

Lemma 3.2. *Let the number of peers in a grid be bounded by $\mathrm{O}(d^2)$. Then the total information that is received during $\mathcal{A}_{GRID}$ by any peer in the grid, is bounded by $\mathrm{O}(d)$.*

Proof. The number of rows, and thus the number of peers in a column is restricted by $\mathrm{O}(d)$, since the number of peers in the grid is bounded by $\mathrm{O}(d^2)$. The received information contains the following messages, their lengths all bounded by $\mathrm{O}(d)$: All IP-addresses of the peers in a row (own row and rows $R-2$, $R-1$) all IPs in a column (own column), the IPs of the extra peers and joining peers and finally the grid positions where peers left the system. □

3.3 Updating Inter-Node Connections

As stated above, the matching between neighbored nodes has to be updated. This is done with the here presented algorithm $\mathcal{A}_{UPDATE}$, which will be run after the grid maintenance algorithm. So the information that is known in $\mathcal{A}_{GRID}$ can be used here, i.e. the information, which peers newly joined the core of the node. The simplest idea to update the matching, is that core nodes send all IPs of the current core, together with the corresponding column numbers to the matched core peers in the neighbors, and in the next round, the received information could be sent to all core peers of the node. The problem with this solution is, that a core peer would receive $\Theta(d)$ core peers from

$\Theta(d)$ neighbors. So the information a core peer would receive would be in $\Theta(d^2)$, which is not desirable. For this, we only send the update information and afterwards take into account possibly missing information. Note that a core peer never changes it's column number in the grid (unless the dimension is changed), but stays on it's place till it leaves the system. Also for $\mathcal{A}_{UPDATE}$ we give a detailed algorithm in rounds. It is run synchronously in all core peers of all grids in the system.

ROUND 1

Outline: All (core) peers of a node know the IPs of the core peers, that joined the core in the last run of $\mathcal{A}_{GRID}$.

Sent Messages: Each core peer at $G_v[x, 0]$ that has not newly joined the nodes core, sends the IPs of the peers, that newly joined the node's core, to it's matching partners $G_{\rho_i}[x, 0]$ for all $2 \leq i \leq n$. Together with the IP, also the column number c of each new core peer is provided as well as the node ID of v, namely $(\ell_1 \ldots \ell_d)$.

ROUND 2

Outline: The message received in the last round is forwarded to the destination.

Sent Messages: Each core peer $G_v[x, 0]$, that received a message in the last round, sends the following information: For each received triple $[IP, c, ID]$, the IP, together with the corresponding node ID is forwarded to $G_v[c, 0]$.

ROUND 3

Outline: The previous core nodes now all know their new matching partners and the corresponding node ID to which they belong. However, a peer newly entered to a core, does not know it's matching partners, that were not replaced in the last run of $\mathcal{A}_{GRID}$.

Sent Messages: Each core peer $G_v[x, 0]$ that received new matching partners $G_{\rho_i}[x, 0]$ in ROUND 2 sends its own IP-address together with its node ID $(\ell_1, \ldots, \ell_n)$ to all its new matching-partners $G_{\rho_i}[x, 0]$.

As already mentioned, $\mathcal{A}_{EDGE}$ will run after $\mathcal{A}_{GRID}$. It will be run on all nodes of P_d at the same time. The following lemma states, corresponding to Lemma 3.1, that the two algorithms can defend the ongoing churn bounded by $d/2$ joins and leaves in the pancake system during 7 rounds that are needed for both algorithms.

Lemma 3.3. *Given a pancake system P_d, that is initially fully repaired. In each node, the two algorithms $\mathcal{A}_{GRID}$ and $\mathcal{A}_{EDGE}$ are run continuously in this order. Let the number of joins and leaves in any time interval of 7 rounds be bounded by $d/2$, and further the number of peers in any node be always at least $2d + 2$. Then the following statement holds: The pancake system after after every run of the two algorithms is fully repaired up to the joins and leaves that occur during the algorithms.*

Proof. We first state that the pancake graph P_d is always fully repaired up to $d/2$ joins and leaves after each run of the two algorithms. This follows (1) from Lemma 3.1, (2) the fact that the same number of joins and leaves now is allowed for the time both algorithms are running and (3) that $\mathcal{A}_{EDGE}$ does not affect the grids. This implies that for any two consecutive full runs of $\mathcal{A}_{GRID}$ and $\mathcal{A}_{EDGE}$, there exists a column number c, for which in each grid of P_d the column c in the system are complete and not changed up possible joining nodes in the top rows. The existence of the complete columns guarantee the that the core peers in column c have all information for the sending of the IPs, column numbers and node ID's to the neighboring core peer in column c in ROUND 1 and the sending to the destination in ROUND 2. Thus all peers that not newly joined the core receive the information about their new matching partners. The sent messages inROUND 3 guarantee, that also new core peers receive the IPs and node ID's of their matching partners. □

Lemma 3.4. *The total size of information that is received by any peer in the pancake system during the run of* A_{UPDATE}*, is bounded by* $\mathrm{O}(d)$.

Proof. The number of sent IP-addresses in the first two rounds is bounded by the number of peers that were replaced in the grid algorithm, thus $d/2$. In ROUND 3, the number of received IP-addresses is bounded by the number of joins and leaves as well. These are obviously in $\mathrm{O}(d)$. □

3.4 Expansion

When the pancake graphs order is increased from d to $d+1$, each node v must split into $d+1$ new nodes (c.f. Chapter 2.1). How can this expansion be achieved on the grid level? As has been mentioned before, the grid G_v consists of $d+1$ columns. This allows a simple way to split the grid: Every column yield one new node.

We know from Chapter 2.1 that two neighboring expanded nodes have already been adjacent in P_d (or originate from the same node). Now assume that two columns G_v and G_u of two expanding adjacent nodes $u, v \in P_d$ become neighbors in P_{d+1}. With the grid as described so far, these two columns have only one connection to each other.

In order to increase the fault-tolerance, the following mechanism is applied: As soon as a certain threshold in the information aggregation algorithm is achieved, which guarantees at least $d+1$ complete rows in each node, the nodes start to establish a matching between the columns in G_u and G_v that will be future neighbors. 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.

The following algorithms will be run after $\mathcal{A}_{GRID}$ and $\mathcal{A}_{EDGE}$, in the same phase as described in Chapter 3. For this we can assume that the joins and leaves of the last phase are fully repaired, and during the run of the following algorithms, only $d/2$ joins and leaves have to be taken into account. Thus in the following, we assume to have a fully repaired pancake graph, and allow $d/2$ joins and leaves during the algorithm runs. Note that it makes no difference, if some of the joins and leaves may occurred before the start of the algorithm.

3.4.1 Matching Establishment

Again, we provide a detailed description of the expansion rounds. First, we describe how to establish the matching, which is done by $\mathcal{A}_{MATCH}$. This is done in d phases:

in phase dim for the matching to neighbor $\rho_i(v)$, which is indicated by the argument i. The idea is that each peer at $G_v[y, y+1]$ with $0 \leq y \leq \lceil d/2+1 \rceil$ sends its IP addresses of its row to the peer $G_v[y, 0]$. Peer $G_v[y, 0]$ is then responsible to transfer the $y+1$-th row to the corresponding peers $G_{\rho_i(v)}[y, 0]$ for $i \in [2, d]$. From there, the information is broadcast to $G_{\rho_i(v)}[\cdot, y+1]$. Despite the adversarial churn, we will be able to guarantee, that there is at least one complete row that reaches its destination. We will describe phase i. We will use i as the argument of the algorithm, thus phase i is referred to as $\mathcal{A}_{MATCH}(i)$. We assume, that the algorithm is run on a fully repaired pancake system, allowing $d/2$ joins and leaves during the algorithm run (c.f. Chapter 3.1).

ROUND 1

Outline Each peer in the grid knows the IP-addresses of its row neighbors. Each peer at position $G_v[y, y+1]$, $0 \leq y \leq (d+1)$, starts the transfer of it's row information to the neighboring node.

Sent Messages: A peer at $G_v[y, y+1]$ sends the IP-addresses of it's row neighbors to $G_v[y, 0]$, the core peer in its column.

ROUND 2

Outline The information is sent to the neighboring core.

Sent Messages: $G_v[y, 0]$ forwards the message received in this round to $G_{\rho_i(v)}[y, 0]$, its matching partner in the neighbor node, given by a prefix-inversion of length i.

ROUND 3

The message is forwarded to the $y+1$-th row.

Sent Messages: $G_{\rho_i(v)}[y, 0]$ forwards the message to $G_{\rho_i(v)}[y, y+1]$.

ROUND 4

Sent Messages: $G_{\rho_i(v)}[y, y+1]$ sends the received messages to$G_{\rho_i(v)}[\cdot, y+1]$.

ROUND 5

Outline: Now the matching, which may not all nodes received is repaired.

Local Computation: Each peer v of column $c$can compute to which column in $\rho_i(v)$ it has to be matched: For $c \geq i$, a matching column k in $G_{\rho_i(v)}$is given by $k = j$. On the other hand, for $c \leq i$, the to be matched column is given by $k = i - c$. Note that in the case $c = i + 1$, two columns have to be matched.

Sent Messages: Each peer that received a row of the neighbors node, sends an ordered array of IPs of the lowest $d+1$ members of it's column to the received peer IPs that belong to columns to which a matching has to be established.

ROUND 6

Outline: The matching establishment is finished.

Sent Messages: The messages that are received, are forwarded into the column.

The following lemma states that the establishment of the matching is guaranteed when there are at most $d/2$ joins and leaves in the system P_d. The algorithm $\mathcal{A}_{MATCH}$ will be run after $\mathcal{A}_{GRID}$ as described in Chapter 4. For this, we can assume that there are at most $d/2$ joins and leaves.

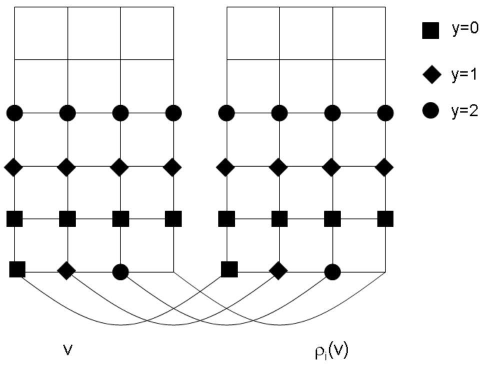

Figure 3.1: Disjoint sets of used peers for different y-values in (P_4).

Lemma 3.5. *Assuming all nodes in P_d have at least $d+1$ rows and P_d is fully repaired up to $d/2$ joins and leaves. After running $\mathcal{A}_{MATCH}(i)$ in all nodes synchronously, the following statement holds: All matchings to $\rho_i(v)$, that are needed for expanding the grid are established up to $d/2$ joins and leaves.*

Proof. We send $\lceil d/2+1 \rceil$ messages that contain $d+1$ IPs. The set of peers corresponding to the IPs, as well as the pathes over which the IPs are sent are disjoint: the IPs of row $G_v[\cdot, y+1]$ are sent to the row $G_{\rho_{dim}(v)}[\cdot, y+1]$, using the core peers $G_{(v)}[y,0]$ and $G_{\rho_{dim}(v)}[y,0]$. We use $\lfloor d/2+1 \rceil$ different y-values. For two different y-values, all involved nodes are obviously disjoint, as long as $y \neq 0$. For this, see also Figure 3.1 Since the grid is repaired up to the at most $d/2$ joins and leaves, for at least y-value $(0 \leq y \leq \lceil d/2+1 \rceil$ the following statement holds: For each node v in P_d, the row $G_v[\cdot, y+1]$ as well as the core peer $G_v[y,0]$ will not leave during the full run of the algorithm. In other words there exists a row, for which the message could be sent in both directions and both rows are complete during the whole run of $\mathcal{A}_{MATCH}$. Such a row number exists, since the number of used rows is $\lceil d/2+1 \rceil > d/2$. A node in this row can now send the columns to the columns that have to be matched. Vise versa, the corresponding IPs and row numbers of the to be matched column can be received and forwarded to the peers that will be their matching partners. Then the matching is complete up to the joins and leaves during the current phase. To conclude, it has to be shown that the computing of the matching partners is correct, i.e. any two columns that establish a matching are indeed future neighbors, and all future neighbors are indeed matched. From symmetry it then obviously holds, that if a column c_1 decides to establish a matching to a column c_2 in a neighbored node, also c_2 takes the decision to match c_1.

In $\mathcal{A}_{MATCH}(i)$, matchings between columns in the nodes $v = (\ell_1 \dots \ell_d)$ and $\rho_i(v) = (\ell_i \dots \ell_1 \ell_{i+1} \dots \ell_d)$ have to be established. The future ID's of the columns have all literals $\ell_1 \dots \ell_d$ in the same order, and additionally, for the c-th column, the new number $(d+1)$ inserted at position $c+1$. For two columns in neighbored nodes to be neighbors, the prefix-inversions must have the same effect on $(\ell_1 \dots \ell_n)$ as the prefix-inversion of their parent nodes. From the position of $(d+1)$ then the column number of the matching partner is easily found. Let now first $c > i$. The literal $(d+1)$ being at position $c+1$, the future nodes are connected by the same prefix-inversion as their current node, not affecting the number $(d+1)$. It follows that in both columns, the entry $(d+1)$ is at the same position and they thus have also the same column number. For $c < i+1$, $(d+1)$ will be part of the prefix-inversion. The future neighbors thus are connected by a prefix-inversion of length $i+1$. A prefix-inversion of length $i+1$ changes the position of any literal from $c+1$ to $i-c+1$ and thus corresponds to the column number $i-j$. Finally, for $i = c$, prefix-inversions of the length i and $i+1$ respectively, affect the literals $\ell_1 \dots \ell_n$ in the same way, and thus result both in a to be matched column, at position j and $i-j=0$ respectively. Note that for the columns $c = 0$ and $c = 1$, $c = i$ never holds. The reason is, that these columns are future neighbors and thus have to establish a matching to only $d-1$ columns in neighbored nodes. □

Lemma 3.6. *The total information that is received by any node during the run of $\mathcal{A}_{MATCH}$ sent is bounded by* $\mathrm{O}(d)$.

Proof. Before the last round, the sent and received messages are obviously all of length $\mathrm{O}(d)$: A single row is received and/or sent in the first four rounds, in ROUND 5 $d+1$ IPs, i.e. the lowest $d+1$ column members, are sent. For round 6 we see that the number of matchings that one column in G_v has to establish to $G_{\rho(v)}$ is 1 or 2 and thus in $\mathrm{O}(1)$, as seen from the calculation in ROUND 5. It follows that the number of peer IPs that have to be sent and received in the column are in $\mathrm{O}(d)$. □

Since we want the out-degree in our pancake system be bounded by $\mathrm{O}(d)$, this also is shown for the complete matching between the columns as stated in the following lemma:

Lemma 3.7. *The total information needed for the matching between columns that will be future neighbors, is bounded by* $\mathrm{O}(d)$.

Proof. In the expanded (future) pancake graph P_{d+1}, each node will be neighbored to d neighbors. Thus each column will be matched to at most d columns in different nodes. A node holds for each of its d (or $d-1$ for column 0 and 1) matched columns one IP-address. Thus the information needed for the complete matching between future neighbors is bounded by d IPs and thus by $\mathrm{O}(d)$. □

3.4.2 Matching maintenance

The matching maintenance algorithm $\mathcal{A}_{MM}$ is based on the same idea as $\mathcal{A}_{EDGE}$. It updates the matchings that were already established by $\mathcal{A}_{MATCH}$. Remember that after the run of $\mathcal{A}_{GRID}$ all peers know their column neighbors. Also the peers that have not newly come into the column, know which peers newly came into the column. For this, the same algorithm as $\mathcal{A}_{EDGE}$ can be used, up to the fact that instead of the matching between core rows, the matching between columns are updated. Note that we have the same number of peers as in the core (we take the $d+1$ lowest rows in each column). So also all proves can be overtaken from Chapter 3.3 since also the maximum number of leaves that have to be repaired are the same as in $\mathcal{A}_{EDGE}$, namely $d/2$. Also for the matching maintenance, we introduce an argument i: In $\mathcal{A}_{MM}(i)$, the matchings to the future neighbors in the nodes $\rho_j(v), v \leq i$ are updated.

Lemma 3.8. *When the number of complete rows in each grid is at least $d+1$, the already established matchings are repaired by $\mathcal{A}_{MM}$ up to the churn in the current phase.*

Proof. See proof of Lemma 3.3 □

Lemma 3.9. *The total size of information that is received and/or sent is bounded by* $\mathrm{O}(d)$.

Proof. See proof of Lemma 3.4 □

3.4.3 Expand Step

In order to change the order of the pancake system from d to $d+1$, the following algorithm $\mathcal{A}_{EXP}$ is used. For the latter, 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 $v_{(i)}^{exp} = l_1...l_{i-1}(d+1)l_i...l_d$.

ROUND 1

Outline: The peers of the i-th column $G_v[i, \cdot]$ which will form the new node $v_{(i)}^{exp}$ are completely connected. Thus, it is easy to build the new grids $G_{v_{(i)}^{exp}}$, including cores and inter-connections: The inter-node connection will consist of the established matching between the former columns plus one additional, to be defined connection, since in the new dimension $d+1$, there are $d+2$ core-peers.

Local Computation: Each peer in $v_{(i)}^{exp}$ locally computes the form of $G_{v_{(i)}^{exp}}$, for example depending on the IP addresses or the row numbers. The peers that were matching peers, namely the peers in the lowest $d+1$ rows are assumed to be new core peers, where the row number in the old grid corresponds to the column number in the new grid. Thus one new core peer has to be computed.

Sent Messages: All new core peers send the IP of the core peer in column $d+2$ together with the corresponding node ID to their matching partners.

ROUND 2

Outline: The expansion is finished.

Sent Messages: The core peers in $v_{(i)}^{exp}$ inform the new core peer about it's matching partners (IP and node ID), received in the last round.

Lemma 3.10. *Let the algorithm $\mathcal{A}_{EXP}$ run on a fully repaired pancake system P_d, and all columns be matched to their future neighbors as described above. Let further the number of complete rows in each grid be at least $2d+4$ and the churn during $\mathcal{A}_{EXP}$ be bounded by $d/2$ joins and leaves. Then the following statement holds: After the run* *of $\mathcal{A}_{EXP}$, the pancake system is of dimension $d+1$ and fully repaired up to the joins and leaves during the expand step.*

Proof. First we note that in the beginning the number of connections between any two new nodes es $d+1$. Thus the $d/2$ leaves can not destroy the connection. The only thing to show is that the computation can be done unambiguously. The argument is the same as for $\mathcal{A}_{GRID}$: An order can be defined as well on the new grid positions and the IP-addresses. $\square$

Lemma 3.11. *The information received during $\mathcal{A}_{EXP}$ is bounded by* $\mathrm{O}(d)$.

Proof. For each neighbor, one IP-address is received, namely the IP of the peer that will be the new core peer. Since in P_{d+1} each node has d neighbors, this is bounded by $\mathrm{O}(d)$. $\square$

3.5 Reduction

If the pancake graph reduces its order from $d+1$ to d, $d+1$ grids have to be merged to one and some nodes have to be interchanged between the dominators (cf. Chapter 2.1). For this, we will use the algorithm $\mathcal{A}_{RED}$. We will again describe the operations of $\mathcal{A}_{RED}$ in terms of the grid representation.

Similarly to the notation introduced in Chapter 2.1, let $v_{(1)}^{dom} \in P_{d+1}$ be the dominator of a cluster that reduces to $v \in P_d$ and let $v_{(i)}^{dom} = \rho_i(v_{(1)}^{dom})$. In order 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 these nodes into one node v (cf. Chapter 2.1).

On the grid level, a constant number of rounds is needed for this order reduction which of course are again assumed to execute concurrently to the adversary.

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). 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 that will form v after the node exchange, $v_{(1)}^{exp}$ being the dominator. Node $v_{(1)}^{exp}$ now 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 simple rearrangement which is not described further here.

In the following we give the detailed algorithm. Note that some rounds in the description could be parallelized. However, in order to enhance clarity, we accept some additional rounds.

ROUND 1

Outline: In order to exchange the dominated nodes according to exchange of the dominated nodes, $v_{(i)}^{dom}$ for $i \in [3, d+1]$ gets the information about the core of its future dominator. Moreover, the latest joiners are integrated into the grid.

Sent Messages: Each peer in $G_{v^{dom}_{(i)}}[x, 0]$ sends the IP address of the corresponding dominator peer $G_{\rho_i(v^{dom}_{(i)})}[x, 0]$ (given by the matching) to $G_{\rho_{i-1}(v^{dom}_{(i)})}[x, 0]$. Additionally, each peer in the system broadcasts its own address and the addresses of its joiners within its row.

ROUND 2

Outline: The to be transferred nodes inform their new dominators about themselves. Nodes start building a clique.

Sent Messages: The core peers $G_{v^{dom}_{(i)}}[x, 0]$ for $i \in [3, d + 1]$ send their addresses to the dominator peers they learned in the previous round. In order to establish a clique, each peer broadcasts the addresses of its row to its column.

ROUND 3

Outline: Inside each node, a clique is built.

Sent Messages: Each peer sends the addresses collected in the previous rounds to their row.

ROUND 4

Outline: Dominated nodes learn about all peers in their dominator.

Sent Messages: Each peer in the core of a dominator, i.e., $G_{v^{exp}_{(1)}}[x, 0]$, sends all addresses in the node to the neighboring peers in $G_{v^{exp}_{(i)}}[x, 0]$ for $i \in [2, d + 1]$.

ROUND 5

Outline: The information about the peer in the neighboring dominator is disseminated inside dominated nodes.

Sent Messages: $G_{v^{exp}_{(i)}}[x, 0]$ for $i \in [2, d + 1]$ distributes the dominator's peer addresses within $G_{v^{exp}_{(i)}}$.

ROUND 6

Outline: The grids reduce to d rows and one column (future column of the reduced node). Thus, each grid position may have a constant number of peers. These d rows then establish a matching to the corresponding row of the dominator.

Local Computation: Each peer in $G_{v^{exp}_{(i)}}[x, y]$ for $i \in [1, d + 1]$ can compute its position in the new grid of d rows (uniformly distributed).

Sent Messages: In order to establish the complete bipartite matching to the peers at the corresponding row of the dominator, the peers in row x of $G_{v^{exp}_{(i)}}$ for $i \in [3, d + 1]$ send their addresses to the peers in row x in $G_{v^{exp}_{(1)}}$.

ROUND 7

Outline: Using the information received in the previous round, the peers in the dominator transform the $d+1$ grids $G_{v_{(i)}^{exp}}$ for $i \in [1, d+1]$ into one grid G_v having at least $d/2$ complete rows.

Sent Messages: Each peer in row x of $Gv_{(1)}^{exp}$ forwards the received packets to all rows x of the nodes $Gv_{(i)}^{exp}$ for $i \in [2, d+1]$.

ROUND 8

Outline: Due to the ongoing churn (e.g., crashed mediating peer in dominator), some peers may not have received the row information. This issue is tackled next.

Sent Messages: Each peer at $G_v[x, y]$ sends a packet to $G_v[x, \cdot]$ saying whether it has received the row information or not, and the total number of peers in its row.

ROUND 9

Outline: Incomplete rows are merged together, forming at least $3d/4$ complete rows. The new core is built.

Local Computation: The new core is computed locally. For each column (former node), the new core peer is given by the previous core peer, that has the lowest IP-address.

Sent Messages: Peers in complete rows send the necessary neighbor information to the peers in incomplete rows in order to integrate them. Core addresses are sent along the columns.

ROUND 10

Outline: The new core is made public within the whole node.

Sent Messages: The core addresses are sent along the rows.

ROUND 11

Outline: Now the inter-connections between the nodes in P_d are established. For this, we kept our connections to the transferred nodes in the exchange of the dominated nodes.

Sent Messages: The old core nodes of $v_{(i)}^{exp}$ for $i \in [3, d+1]$ send the core of the new node v to their previous neighbor $\rho_{i-1}v_{(i)}^{exp}$. Note that since all nodes $v_{(i)}^{exp}$ originally had a different dominator, this procedure yields all necessary connections.

ROUND 12

Outline: The information about the core is propagated.

Sent Messages: The received core information is broadcasted inside the column.

ROUND 13

Outline: The establishment of the inter-connections is continued.

Sent Messages: Core information is forwarded to the corresponding column.

ROUND 14

Outline: The new connections established.

Sent Messages: Core information is sent to the core peers in order to complete the matching.

ROUND 15–16

Outline: We now start to repair the grid. Up to the joins and leaves during this algorithm, it will be fully repaired at the end of the algorithm.

Sent Messages: Each peer sends the number of peers in its row and the number of peers in its column to all neighbors (taking 2 rounds), which allows every peer to compute locally the state of the grid. Moreover, each peer in row y sends all IP addresses of its row to its adjacent peer in row $y + 1$. In order to handle faults, this information is then also broadcasted along the row in ROUND 16.

ROUND 17

Outline: The number of peers per row is made a multiple of $d + 1$ in order to split rows later.

Local Computation: Each peer computes how many peers of each row have to be sent to another row. Since each row contains at least $d + 1$ peers, the idea is that peers are only transferred from a higher row $y + 1$ to the next lower row y. The computation works as follows: row 1 sends so many (uniquely defined) peers to row 0 such that the number of peers in row 0 is divisible by $d+1$. This procedure is then repeated between row 2 and row 1, and so on. The assigning of new column numbers to the peers are done, such that the number of peers in the row that have to change the column is minimized.

Sent Messages: The IP addresses of the peers that have to move are sent to the lower row.

ROUND 18

Outline: Peers are distributed evenly among the rows.

Local Computation: Given the number of peers at each position, we can locally compute which peer has to move to which place. The remaining peers in the top row, are assigned to the lowest columns, one remaining peer in each column. This will be the extra peers.

Sent Messages: Moving peers are provided with the necessary neighbor information.

ROUND 19

Outline: If a peer has not received its new column information, it informs its previous column about its new column.

Sent Messages: If necessary, a peer informs its old column as described above.

ROUND 20–22

Outline: The requests of the previous rounds are satisfied.

Sent Messages: The information, which peer requested information about which column is broadcasted in the column. Then, then this is sent to the column of which the information is missing. Finally, the column of the peer receives the column information from its new column.

ROUND 23

Outline: Some transferred peers may still be unknown to other transferred peers.

Sent Messages: Addresses of transferred peers are broadcasted along the column.

ROUND 24

Outline: The rows are finally split. The grid is now repaired up to the churn that happened during the execution of this repairing phase.

Local computation: The peers can based on the IP-addresses decide, which neighbor will be part of which row. Of course the core peers, known by all peers in the grid are assigned to row 0.

Lemma 3.12. *Let $\mathcal{A}_{RED}$ run on a fully repaired pancake system P_{d+1}. Let the churn during the run of $\mathcal{A}_{RED}$ be bounded by $d/2$. Then the following statement holds: After the run of algorithm $\mathcal{A}_{RED}$ on all grids of the nodes of P_{d+1}, the result is a pancake graph of order d, which is fully repaired up to the churn that happened during the run of the algorithm.*

Proof. We first note that the pancake system is during the whole run of $\mathcal{A}_{RED}$ fully repaired up to $d/2$ joins and leaves with respect to the old dimension. This guarantees that no connections between nodes (i.e. the inter-node connections) are completely lost. Moreover, there exist at least $\lceil d/2 + 1 \rceil$ column numbers in the old graph P_d, for which all corresponding columns in the grid are complete during the whole run of $\mathcal{A}_{RED}$. To enhance clarity, the proof is done round by round. After ROUND 1 it is guaranteed by the observation above, that each node $v_{(i)}^{dom}$ receives a matching of its core to its dominators' core. Since the diameter of the grids in P_d is 3, which is guaranteed by the complete columns, the cliques are established in ROUND 4. The IPs of the dominators' nodes received in ROUND 5 are thus received by all peers in a non-dominator node in ROUND 6. Again, the computation in ROUND 6 can be done unambiguous, using an order on the IP-addresses of both the dominators peers and the peers in the non-dominators node. These IPs are all known by each peer of the node. The new (temporary) grid will consist of d rows. Since each old node consist of at least $2d + 2$ peers, and each old node distributes the peers equally over the new grid rows (up to rounding errors), at least 2 peers are assigned to each grid position. Taking in account the up to $d/2$ leaves during the run of $\mathcal{A}_{RED}$, at least $3d/4$ rows will have at least one surviving peer on each grid position. This observation guarantees, that at least $3d/4$ rows can be established after round 7 (and will survive as long as needed in $\mathcal{A}_{RED}$). Also at least $d + 1 - d/4 = 3d/4 + 1$ columns are complete, again yielding a grid diameter of at most 3. For this, also the messages in ROUND 9 can be sent and will be received, based on a unambiguous computation that assigns each IP-address in an incomplete row to a complete row. Since in each column a surviving core peer exists, it can be computed in ROUND 9. With the guaranteed diameter, all peers know the IPs of the core of the new node after ROUND 10. The establishment of the inter-node connections in rounds ROUND 10–14 can be guaranteed, since each peer in the node knows the core peers in the node and the connections to the previous neighbors are not lost as stated above and again by the diameter of the grid. The latter also guarantees the successful sending in ROUNDS 15–16. The computation in ROUND 17 can be done unambiguous by the above: Each peer knows all peers in its row as well as their grid positions and the number of peers in the other rows. From this, the IPs of the peers that will leave and join the column can be computed. The receiving of the information after ROUND 22 is guaranteed, since there still exist complete columns in the grid and for ROUND 23 it is sufficient that each column has members that have not changed the column in the last rounds, which obviously holds (see Local Computation in ROUND 17). Finally, in ROUND 24, the peers are evenly distributed among the rows and columns by the above, i.e. the computations and the guaranteed receiving of the messages. □

Lemma 3.13. *Assume the number of peers in each node of P_d is bounded by $\mathrm{O}(d)$. Then the total information that is received by any node in algorithm $\mathcal{A}_{RED}$ is bounded by $\mathrm{O}(d)$.*

Proof. Since the old nodes are assumed to be bounded by $\mathrm{O}(d+1)$, all IPs of a constant number of old nodes can be sent (i.e. the dominator nodes in each cluster) and an old node can build a clique in $\mathrm{O}(d)$. A new column in P_d consists of $d + 1$ old nodes, thus bounded by $\mathrm{O}(d^2)$. Note that from the beginning, the new grid consists of $\Theta(d)$ rows, which are equally distributed up to rounding errors, bounding the row information by $\mathrm{O}(d)$. Beginning in ROUND 10, the new inter-node connections are established: The procedure as described above, guarantees that (1) each previous node receives at most one neighbored complete core ($d + 1$ IPs) and (2) the received core-peers IPs (and corresponding node ID's) are, after broadcasting them in the column, directly sent to the right column. Thus in a column, the peers receive the own core, one complete neighbors core, and finally for each neighbor one core-peer. In the round 17 to 24, the received messages include for each column the number of peers, the IPs of one complete row, the IPs of one complete column as well as the number of peers in each column and row. All bounded by $\mathrm{O}(d)$ as shown above. □

3.6 Token Distribution on the Grid Level

Now the token distribution on the grid level is presented. As in the whole text, the number of rounds will not be minimized. We only want to assure that the number of needed rounds is constant and not dependent on d. The token distribution algorithm of Chapter 3.6 can not be used directly: Since our out-degree is assumed to be in $\Theta(d)$, it is of course not desirable that an information amount of $O(d^2)$ has to managed in constant time. However, we can not guarantee that this is not the case, when all transferred peers within a cluster have to pass the dominator node: the number of neighbors is $O(d)$, and also the number of peers that have to be sent between two neighbor are bounded by $O(d)$. Since we cannot decide which peers may leave the system during the algorithm, the load can not be balanced directly between the core peers.

As described later in Chapter 4, the token distribution algorithm is run after the grid maintenance algorithm. For this, the number of peers in a node (up to joins and leaves in the current phase) and the IP-addresses of all extra peers are known by all peers in a node. In this chapter, with the number of peers in a grid, the number of peers of the grid at the end of the previous phase is meant. Since in every round, some peers may join and leave, we never know the exact current number of peers in the system. However, the error corresponds at most to the allowed number of leaves or joins in the system. We do here not define explicitly the dominators. We just assume that they are well defined. However, this could be easily done. For example in phase i, each node $v = (\ell_1, \dots, \ell_d)$, for which $\ell_1 = max\{\ell_1, \cdots, \ell_{i-1}\}$ could be defined as a dominator.

ROUND 1

Sent Messages: All core peers send the number of peers in their grid to their matching core peer in $\rho_i(v)$.

ROUND 2

Local Computation: The core peers compute the difference of the number of peers, i.e. $|v| - |\rho_i(v)|$. If $|v| > |\rho_i(v)|$, the number q_1 of the to be transferred peers is computed: $q_1 = \lfloor |v| - |\rho_i(v)| \rfloor$

Sent Messages: If $|v| > |\rho_i(v)|$, the core peers send the number of the to be transferred peers into their column.

ROUND 3

Sent Messages: All peers send the information received in the last round to all row neighbors.

ROUND 4

Outline Now all peers in the node are informed about the number of peers that will be sent to the node $\rho_i(v)$. If $|v| > |\rho_i(v)|$, the new number of rows is computed. The IP-addresses of the peers in the vanishing rows will be broadcasted in the grid.

Local Computation Every peer computes the new number of rows R_{new} of the node, based on the current number of rows R_{old} of the node: $R_{new} = \lfloor R_{old} - \frac{q_1 - e}{d+1} \rfloor$. e is the number of extra peers in the grid.

Sent Messages: Each peer having a row number larger than $R_{new} - 1$ sends the ID's of its row to its column neighbors.

ROUND 5

Outline This round guarantees that in the next round, all peers will receive the IPs of the peers in the rows $R_{new}...R_{old} - 1$.

Sent Messages: The information received in the last round is forwarded to the row neighbors.

ROUND 6

Outline All peers know all IP-addresses of the peers that are in the rows larger then $R_{new} - 1$. The peers that are sent to the neighbor $\rho_i(v)$ can be computed locally (lowest IP-addresses). Note that the peers were provided earlier with the number of peers that have been sent to the neighbor. All remaining peers can compute the remaining peers, i.e. those their IP'addresses have been forwarded, but are not transferred to the neighbor.

Local computations Each peer computes which peers are sent to the neighbor node $\rho_i(v)$ (lowest q IPs). For reasons seen in round 10, all peers set $R_{old} := R_{new}$.

Sent Messages: The core peers send the ID's of the to be transferred peers to their neighbored core peers in $\rho_i(v)$. All core peers in the system send the number of peers in their node *after* the exchange to their dominator.

Remark The remaining peers in the senders node, i.e. the peers that were prepared for sending but not sent, are also called extra peers in this node. Also, in the receivers node, the set of extra peers is referred to the extra peers at the beginning of the algorithm plus the peers that were received in the exchange with $\rho_i(v)$.

ROUND 7

Outline The cores of all nodes are provided with the IPs of the joining peers. Moreover, the dominator knows for each node in it's cluster, of how many peers it consists. Based on this, the dominator will decide how many peers are sent between which nodes.

Local Computations: The dominator computes the peers exchange within it's cluster. The way how this is computed is not described here. It is obviously possible to do this in such a way, that at most $\mathrm{O}(d)$ tokens have to be exchanged between any two nodes. i.e. The number of to be sent or received peers is bounded by $\phi/2 = 4d + 3(J + L) = 7d/2$ for each node.(c.f. Theorem 2.7)

Sent Messages: Each core peer in the dominator node, sends a message to all neighbors that have to send nodes with the following information: For each sending that has to be done the number of peers to be sent, and the IP-address of the matching peer of the destination, that is, the connection to the destination peer.

ROUND 8

Outline: The core peers in the cluster are informed, how many peers have to be sent to which node and additionally, the corresponding core peers are known. The sending of the peers is prepared: When the number of to be sent peers exceeds the number of extra peers, new peers are prepared for sending, similar to the first peers exchange.

Local Computations: The core peers calculate the difference $\delta = q_2 - e$, between the to be sent peers q_2 and the current number of extra peers e.

Sent Messages: In all nodes, where $delta > 0$, t he core peers send δ, into their column.

ROUND 9

Outline This Round corresponds to round 3:

Sent Messages: All peers send the information received in the last round to all row neighbors.

ROUND 10

Outline This Round corresponds to round 4: All peers in the node are informed about δ. If $\delta > 0$, the new number of rows is computed. The IP-addresses of the peers in the vanishing rows will be broadcasted in the grid.

Local Computation Every peer computes the new number of rows, based on the current number of rows R_{old} of the node: $R_{new} = \lfloor R_{old} - \frac{\delta}{d+1} \rfloor$.

Sent Messages: Each peer having a row number larger than $R_{new} - 1$ sends the IPs of its row to it's column neighbors.

ROUND 13

Outline: This round corresponds to round 5: This round guarantees that in the next round, all peers will receive the IPs of the peers in the rows $R_{new}...R_{old} - 1$.

Sent Messages: The information received in the last round is forwarded to the row neighbors.

ROUND 14

Outline: The peers are sent to the neighbors, corresponding to the decision of the dominator.

local computations: The core peers compute which peers are sent to which nodes: The destination nodes can be ordered by their Permutation, and the peers that were prepared for sending by their IP: the lowest IPs are sent to the node with the lowest permutation.

Sent Messages: The IPs of the to be transferred peers are sent by the core peers to the corresponding core peer as computed above.

ROUND 15

Outline: The core peers now have received the IPs of the peers that join the node. These are now broadcasted in the grid, before integrating them.

Sent Messages: The core peers send the IPs of the joined peers into their column.

ROUND 16

Outline The broadcast is completed.

Sent Messages: The information received in the last round is forwarded to the row neighbors.

ROUND 17

Outline: All peers now know the IPs of the peers that are part of the node, but not yet part of a row. The joining peers are now integrated into the grid.

Local Computations: The joining peers are assigned to grid positions as follows: By increasing IPs, each peer is assigned to the lowest possible row, and then to the lowest possible column. Of course, each peer in the grid can now decide which of the joining peers will end up in the same column. Also all peers get to know the IPs of the new extra peers.

Sent Messages Each peer that is not a joiner, sends the column and row information to the peers that will newly be part of the same column. Further, all peers send the column and row peers to the joiners, that are joiners as well. With each IP, the corresponding grid coordinates are provided.

Lemma 3.14. *Let $P_d, d > 1$ be a pancake system that is initially fully repaired. Further, in all core peers, the variable i is set to the value same value $2 \leq i \leq d$. Finally, the number of joins and leaves during the algorithm is bounded by $d/2$. Then the following statement holds: After the run of $\mathcal{A}_{TD}(i)$, the peers are distributed in P_d, corresponding to i-th phase of Algorithm 1. Also, all grids are repaired up to the joins and leaves that may occur during the run of $\mathcal{A}_{TD}(i)$.*

Proof. We again state that in each grid, at any time during the whole run, there is a column c that is not affected by the ongoing churn in all grids. This again guarantees the grid diameter to be at most 3, and that the connection between nodes is not broken. This obviously guarantees the successful sending of all messages: Each sending through a row is received by column c. The sending between the nodes is guaranteed by the core peers $G_v[c, 0]$ in each node. Finally all messages sent through a column succeed are received and can be forwarded in column c and the sending into the rows, are received by all core peers when sent from column c in any grid. Note that the column number c is not known in advance, but it is guaranteed that it exists. The computations are unambiguous. Where this is not obvious, it is pointed out in the algorithm. Finally by construction, as pointed out above, the algorithm corresponds to the algorithm presented in Chapter 2.3. The repairing of the grid in the end is done like in $\mathcal{A}_{GRID}$, which guaranteed that indeed the pancake system P_d is repaired up to the joins of leaves during $\mathcal{A}_{TD}$. □

Lemma 3.15. *The total information that is received by any node during the algorithm $\mathcal{A}_{TD}$ is bounded by* $\mathrm{O}(d)$.

Proof. In ROUND 1 to ROUND 3 only single integer values are sent. Beginning with ROUND 4, all IPs of the to be transferred peers are broadcasted throughout the grid (plus at most d peers of a row, that will not remain complete. However, the to be sent nodes is bounded by $\phi = 4d + 3J + L = 7d$ as seen in Theorem 2.7, and thus in $\mathrm{O}(d)$. For each non-neighbored node to which IPs have to be sent, each core peer receives one IP-address. Since the core peer has only $d-1$ neighbors, also here the received information is bounded by $\mathrm{O}(d)$. The last step is similar to the grid maintenance algorithm: Row and column information are sent to the new peers. □

3.7 Information Aggregation Algorithm in Our System

The information aggregation algorithm is very simple to realize at the grid level. However, for completeness it is described here in rounds. As described in Chapter 3.1, we can rely on grids that are fully repaired up to possible joins and leaves in the current phase. Further, the number of peers in the grid are known, again, up to the leaves and joins in the current phase. It would be desirable, to do the steps for all dimensions in parallel in order to get a new estimation of the total number of peers in the system in each run of $\mathcal{A}_{IA}$. However, we then would have to receive $\Theta(d^2)$ messages in one round: 1 integer value to aggregate the result for P_2, 2 integer values to aggregate the value for P_3 and so on, ending finally with $d-1$ integer values to aggregate the value for P_d. So the number of to be received integer values would be in $\Theta(d^2)$. For this the algorithm is not run in parallel and we only get a result after d runs of $\mathcal{A}_{IA}$. This also has an advantage: since as seen in Chapter 4, $\mathcal{A}_{IA}$ always runs after $\mathcal{A}_{TD}$. Since both algorithms need the same number of $d-1$ runs to complete, the receiving of a new value of $\mathcal{A}_{IA}$ also guarantees that the tokens have been distributed in all subgraphs. We used a variable i, which indicates for which subgraph $P_i(v)$ containing v, the information is aggregated in the current round. The result during the calculation ($i < d$) is stored in the variable $curr$. The final result, that is the result for the whole graph, will then be stored in $estimation$. Again, because of the ongoing churn, it is always possible that a certain message is not received. To solve this problem, in the algorithm, we used a boolean variable $complete$, which is set $FALSE$, whenever a information is not received. To solve this problem, the core peers that have the correct result, thus having $complete == TRUE$ send the result array of the previous phase to all core peers of the node. This also guarantees, that new core nodes can run the algorithm too. Nodes, that newly enter the core row $G_v[\cdot, 0]$, thus also set the value of $complete$ to $FALSE$.

The algorithm consists of 5 rounds. The algorithm as described in Chapter 2.2 is done in ROUND 2,ROUND 3 and ROUND 4. In the first round is the sending of the previous result as just described. The result of the previous run of the algorithm is sent into the core row. In ROUND 2, the result of the last round or, if $i = 2$, the number of tokens at each the node, is interchanged with $\rho_{dim}(v)$, and then forwarded to the neighbors $\rho_j(v), j < i$ in ROUND 3. In ROUND 4, the result of the current round is computed, using the received messages. With $curr(\rho_j(v))$, we mean the number $curr$ that was received from the node $\rho_j(v)$. As stated earlier and pointed out in detail in Chapter 4, the goal of $\mathcal{A}_{IA}$ is that the peers in a node know, if a matching has to be established and/or updated, or the dimension has to be changed. In the above algorithms, we assumed that at the beginning of $\mathcal{A}_{MATCH}$, $\mathcal{A}_{MM}$, $\mathcal{A}_{EXP}$ and $\mathcal{A}_{RED}$ respectively, all nodes were informed that the corresponding algorithm begins. To guarantee this, in ROUND 4 and ROUND 5, the result is sent to all peers in the grid.

The following lemma states, that $\mathcal{A}_{IA}$ can be run in a phase with $\mathcal{A}_{GRID}$ and $\mathcal{A}_{UPDATE}$ and it provides the estimated value for $estimation$ after $d-1$ consecutive runs of the 3 algorithms.

Lemma 3.16. *Let $P_d, d > 1$ be a pancake system that is fully repaired up to $d/2$ joins and leaves and $i = 2$ in all core peers. Let now run the following three algorithms $d-1$ times in this order: $\mathcal{A}_{GRID} \rightarrow \mathcal{A}_{UPDATE} \rightarrow \mathcal{A}_{IA}$. Let be the churn bounded by $d/2$ joins and leaves during each run of these three algorithms. Then the following statement holds: In the, end each peer in P_d that not newly joined during the last run of the three algorithms, has the same value for $estimation$, and $estimation$ corresponds to the total number of peers in the system on beginning of the phase on which the first run was started.*

Proof. We first show, that in the end, for any core peer having $complete = TRUE$, the value of the variable $estimation$ corresponds to the total number of peers that are in P_d at the beginning of the first run of $\mathcal{A}_{GRID}$. Since during each run of $\mathcal{A}_{IA}$, P_d is fully repaired up to $d/2$ joins and leaves, there exist column numbers, for which all columns are complete. Moreover, having $d+1$ columns, there is at least one column number for which all core peers stay alive during 2 runs of the algorithm. This guarantees that there is always one column number, in which the calculation succeeds ($complete = TRUE$), and from which all core peers receive the current result in ROUND 2 of the next run of $\mathcal{A}_{IA}$. The way how the new estimation is calculated corresponds to the description in Chapter 2.2 and correctness was proofed in Theorem 2.3. The above guarantees that the computation can be done completely .Again, since there is a column number for which all columns are complete, the variable $threshold$ can be sent into a complete column in each grid at the end of ROUND 4 and thus is indeed received by all peers in P_d after sending the information into the rows. As described above, only the peers that joined in the current run of $\mathcal{A}_{GRID}$,$\mathcal{A}_{UPDATE}$ and $\mathcal{A}_{IA}$ will not get the result. □

Lemma 3.17. *The total size of information that is received by any node during the algorithm $\mathcal{A}_{TD}$ is bounded by $\mathrm{O}(d)$.*

Proof. In ROUND 1, 2 integer values may be received. In round ROUND 3 and ROUND 4, from the neighbors $\rho_i, i \leq dim$, one integer is received. The number of neighbors in P_d is $d-1$. In the end, only the value of $estimation$ is received. □

Algorithm 2 $\mathcal{A}_{IA}$

```
1:  ROUND 1 (core peers)
2:  if $succeed = TRUE$ then
3:      send $i, estimation, curr$ into the core row
4:  end if
5:  ROUND 2 (core peers)
6:  if $succeed = FALSE$ then
7:      receive $dim, curr$ from any core peer in row
8:  else
9:      $dim = 2$
10: end if
11: $succeed := TRUE$
12: if $dim = 2$ then
13:     $curr = |v|$
14: end if
15: send message $curr$ to $\rho_i(v)$
16: ROUND 3 (core peers)
17: receive message $curr(\rho_i)$ from $\rho_i(v)$
18: $curr := curr + curr(\rho_i)$
19: if message was not received then
20:     $complete := FALSE$
21: end if
22: i=1
23: for $i = 2$ to $i$; $j++$ do
24:     send message $curr(\rho_i)$ to $\rho_j(v)$
25: end for
26: ROUND 4 (core peers)
27: while  do
28:     for $i = 2$ to $i - 1$; $j++$ do
29:         receive message $curr(\rho_j)$ from $\rho_j(v)$
30:         if message was not received then
31:             $success := FALSE$
32:         end if
33:         $curr := curr + curr(\rho_j)$
34:     end for
35: end while
36: if $success = TRUE$ then
37:     if $i = d$ then
38:         $estimation := curr$
39:     end if
40:     send $estimation$ to all column neighbors
41: end if
42: ROUND 5 (non-core peers)
43: all non-core peers: receive $estimation$ from core in column
44: all peers: send $estimation$ to all row neighbors
```

Chapter 4

The System

4.1 Overview

Armed with the algorithms of Chapter 2 we can now put our system together. In Algorithm 3, it is shown how the components presented in earlier chapters can be assembled in order to build the pancake system, a P2P-system resilient to an adversary allowed to remove or join $\Theta(\frac{logn}{loglogn})$ peers in constant time, based on the algorithms presented in Chapter 3. The algorithm is given for $d > 1$. The case of $d = 1$ is not discussed in detail here: It is just a clique. Note that for $d = 1$, more leaves and joins have to be tolerated as $d/2$. This is obviously simply possible. For the latter we omit the starting phase and assume, the algorithm is running and all variables are set correctly. We restrict the concurrent adversary to $d/2$ joins and leaves during each time period, corresponding to the longest possible phase, thus to $\Theta(d)$ in constant time. The longest possible phase is given by first the grid maintenance algorithm $\mathcal{A}_{GRID}$, consisting of 4 rounds, followed by $\mathcal{A}_{EDGE}$ (3 rounds), $\mathcal{A}_{TD}(i)$ (17 rounds), $\mathcal{A}_{IA}(i)$ (5 rounds) and finally $\mathcal{A}_{RED}$, consisting of 24 rounds, ending with a total of 53 rounds. Our adversary is thus allowed to remove or join $d/2$ peers in any time interval corresponding to 53 rounds. Once again, we point out that the number of rounds has not been minimized: For example the token information aggregation $\mathcal{A}_{IA(i)}$ could easily be done in parallel with other algorithms. By not optimizing the number of rounds, a much simpler overview of the system can be given. Moreover, the asymptotically results remain the same. Note that we only need one variable i as the arguments for $\mathcal{A}_{IA}(i)$, $\mathcal{A}_{TD}(i)$ and $\mathcal{A}_{MATCH}(i)$ respectively. The reason for this is twofold: First the number of phases for a complete run of the corresponding algorithm is the same for all mentioned. The second reason is that the information aggregation algorithm is not run in parallel. This guarantees that a new threshold is achieved only all $d-1$ phases. Since the starting of $\mathcal{A}_{MATCH}$ depends on whether a certain threshold is passed, the matching establishment algorithm is run synchronously to the information aggregation algorithm.

In the first 4 lines, the algorithms $\mathcal{A}_{GRID}$, followed by $\mathcal{A}_{EDGE}$, $\mathcal{A}_{TD}(i)$ and $\mathcal{A}_{IA}(i)$ are run. These algorithms are part of any phase. In line 5, the variable $estimation$ is compared with $t_m(d) \cdot d!$. $t_m(d)$ is the threshold that guarantees that at least $d+1$ complete rows in any grid of the pancake system exist. Since the thresholds, as described in Chapter 4.2 are given in peers per node, and $estimation$ estimates the total number of peers in the system, we multiply any threshold with $d!$ before comparing it with $estimation.$ This because the pancake graph consists of $d!$ nodes. For further information on the thresholds, see the corresponding Chapter 4.2. We now assume that $estimation \geq t_m(d)$ and thus the if-clause is entered. The variable $complete$ is set to $TRUE$, when the whole matching is already established. If this is the case, the algorithm $\mathcal{A}_{MM}(d)$ is run, thus the matching is updated. As stated in Chapter 3.4.2, the argument d means that the matching is updated to the future neighbors in all neighbored nodes. If on the other hand $complete = FALSE$ the matchings to neighbors for which a matching already exists are updated by $\mathcal{A}_{MM}(i-1)$, namely the matchings to all neighbors $\rho_j(v), j \leq i-1$. In the case where $i = 2$, no matching has been established yet, and thus nothing is done. Then, in line 10 of the algorithm, the matching to the neighbor $\rho_i(v)$ is established. If now $i = d$, the matchings to the last remaining neighbor $\rho_d(v)$ is established and the variable $complete$ is set to $TRUE$. If $i \neq d$, no further algorithm is run in this phase. In line 16, it is checked if $i = d$. If this is the case, a new result of the information aggregation algorithm is achieved. Then this new result, given by the variable $estimation$, is compared with the thresholds. Also, the variable i is set to 2 (line 17), indicating that the $\mathcal{A}_{TD}(i)$ and $\mathcal{A}_{IA}$ will be restarted with the argument 2 in the next phase. If $estimation \geq t_e(d)$ or $estimation \leq t_r(d)$, the pancake system is expanded to P_{d+1} by $\mathcal{A}_{EXP}$ (line 19) or reduced to P_{d-1} by $\mathcal{A}_{RED}$ (line 22) respectively. Additionally, the variable $complete$ is set to $FALSE$, indicating that in the new graph, no matchings are established yet and thus no matchings have to be updated in the next phase. In Chapter 4.2, we will show that no threshold is achieved before the first result of $\mathcal{A}_{TD}$ is supplied. Finally, $estimation$ is compared with $t_m(d)$, the threshold that indicates if the matchings between future neighbors have to be established. If not, i.e. $estimation \leq t_m(d)$, $complete$ is set to $FALSE$. In line 32, all unnecessary IPs are deleted, that may were saved during the algorithms of the phase. This is necessary in order to guarantee, that only $\mathrm{O}(d)$ memory is needed for the system maintenance in any peer of the system. The information that is not deleted include the variables for the algorithms, the row and column neighbors, for the core peers the matching partners in the neighbors nodes which are necessary to represent the edges, and finally, if $estimation \geq t_m(d)$, the matching partners to the future neighbors, that are already established.

To conclude, it has to be mentioned, that joining nodes also must receive the current variables that are needed for the algorithms. This data can be received from the node, to which a joiner newly connected. The nodes that are interchanged by $\mathcal{A}_{TD}$, receive the result of the information aggregation algorithm in round 4 or 5 of $\mathcal{A}_{TD}$. Interchanged nodes do not become directly core nodes, and thus are not involved in the calculation.

Algorithm 3 System (high-level)

```
1:  $\mathcal{A}_{GRID}$ (c.f. Chapter 3.2)
2:  $\mathcal{A}_{EDGE}$ (c.f. Chapter 3.3)
3:  $\mathcal{A}_{TD}(i)$ (c.f. Chapter 3.6)
4:  $\mathcal{A}_{IA}(i)$ (c.f. Chapter 2.2)
5:  **if** $estimation \geq t_m(d) \cdot d!$ **then**
6:      **if** complete=TRUE **then**
7:          $\mathcal{A}_{MM}(d)$ (c.f. Chapter 3.4.2)
8:      **else**
9:          $\mathcal{A}_{MM}(i-1)$ (c.f. Chapter 3.4.2)
10:         $\mathcal{A}_{MATCH}(i)$ (c.f. Chapter 3.4.1)
11:         **if** $i = d$ **then**
12:             $complete := TRUE$
13:         **end if**
14:     **end if**
15: **end if**
16: **if** $i = d$ **then**
17:     $i := 2$
18:     **if** $estimation \geq t_e(d) \cdot d!$ **then**
19:         $\mathcal{A}_{EXP}$ (c.f.Chapter 3.4.3)
20:         $complete := FALSE$
21:         $d := d + 1$
22:     **else**
23:         **if** $estimation \leq t_r(d) \cdot d!$ **then**
24:             $\mathcal{A}_{RED}$ (c.f. Chapter 3.5)
25:             $complete := FALSE$
26:             $d := d + 1$
27:         **end if**
28:     **else**
29:         **if** $estimation \leq t_m(d) \cdot d!$ **then**
30:             $complete = FALSE$
31:         **end if**
32:     **end if**
33: **end if**
34: **delete** all IPs that are not needed any more
35: $i := i + 1$
36: GOTO 1
```

4.2 Thresholds

The pancake system presented by now is complete up to the thresholds: It has to be shown that it is possible to choose thresholds in $\Theta(d)$ for reducing and $\Theta(d^2)$ for expanding, such that our system can compete an adversary, being able to join and remove $\text{O}(d)$ peers within constant time. The thresholds for expanding and reducing should be in $\Theta(d^2)$ and $\Theta(d)$ respectively, in order to guarantee that the node out-degree within a node is at most linear in d before expanding and after reducing. So our system would preserve asymptotically the out-degree $logn/loglogn$ of the pancake graph P_d. Corresponding to earlier chapter, we assume that the adversary is restricted to at most

$d/2$ joins and leaves during any time interval of 53 rounds, which corresponds to the longest possible phase. Let further kd be the maximum discrepancy between the number of nodes at a peer and the average number of peers per node. The threshold for reducing is referred to as $t_r(d) = c_{r,1}d + c_{r,0}$ and the threshold for expanding as $t_e(d) = c_{e,2}d^2 + c_{e,1}d + c_{e,0}$ peers per node. We need a third threshold in $\Theta(d^2)$, which indicates that the matching establishment has to be started. Let this threshold be $t_m(d) = c_{m,2}d^2 + c_{m,1}d + c_{m,0}$. In order to find the missing constants, we first look at the restrictions for the reduce step which changes the order from $d+1$ to d. When the threshold is achieved, the total number of peers may has changed by $d(d+1)/2$ since the information aggregation algorithm supplies the number of peers in P_{d+1} with a delay of d phases. The next result of the information aggregation algorithm will be supplied $d-1$ phases after the reduction, changing the total number of tokens by at most $d(d-1)/2$. So $d-1$ rounds after the reduce step, the average number of tokens per node is at most $(d+1)\cdot t_r(d+1) + ((d+1)\cdot d + d\cdot(d-1))/(2\cdot d!)$, since each node of the reduced pancake P_d consists of $d+1$ nodes of the previous pancake P_{d+1}. The threshold for establishing the matching should not be achieved already. Since $d+1$ nodes merge to one new node, $t_r(d+1)$ is multiplied by $(d+1)$.

$$(d+1)\cdot t_r(d+1) + \frac{d^2}{d!} < t_m(d) \tag{4.1}$$

The restriction for a following reduce step after reducing is computed by the same idea. Now, the worst case is given by leaves instead of joins. The last term takes into account, that the threshold may not exactly be reached. i.e. it can be passed by at most $(d(d+1)-1)/2d!$ peers per node in P_{d+1}: For each of the d phases in which no result is received from $\mathcal{A}_{TD}$, $(d+1)/2$ peers may have been added to the system. One is subtracted in the denominator, since the threshold was not achieved by the last result of $\mathcal{A}_{TD}$. So as a second restriction we get:

$$(d+1)\cdot t_r(d+1) - \frac{d^2}{2d!} - \frac{d(d+1)-1}{2d!} > t_r(d) \tag{4.2}$$

For the expand step we get another two restrictions. First we have to guarantee, that the threshold $t_m(d+1)$ is not achieved in the first d phases after expansion: After d phases we get the first result of the token distribution algorithm in P_{d+1}. Again, the threshold that gives the decision to increase the dimension of P_d may be passed. In P_d, this can be at most $(d(d-1)-1)/2d!$ peers per node with the same argumentation as above. Note that $t_m(d)$ in the first term is divided by $d+1$, since in the expand step, all nodes are split into $d+1$ new nodes.

$$\frac{t_e(d)}{d+1} + \frac{d^2}{(d+1)!} + \frac{d(d-1)-1}{2d!} < t_m(d+1) \tag{4.3}$$

The restriction given by a possible reducing after expanding is:

$$\frac{t_e(d)}{d+1} - \frac{d^2}{2(d+1)!} > t_r(d+1) \tag{4.4}$$

We further have to guarantee that during a reduce step, enough peers are in each node. We assumed to have at least $2d+2$ peers in each node of P_{d+1} before the reduction to P_d, giving the following restriction:

$$t_r(d+1) - k(d+1) - \frac{d(d-1)}{2d!} - \frac{d(d+1)-1}{2d!} \geq 2d+2 \tag{4.5}$$

for some constant k. Beside the threshold $t_r(d)$, the left hand side includes the maximum discrepancy in number of peers per node from the average, the number of possible leaves during the delay given by $\mathcal{A}_{IA}$ and the value the threshold may be passed. The last restrictions are object to the expand step. An expansion can only be done, when a matching has been established. To guarantee this, $d-1$ phases are needed between the start of the matching establishment and the expansion step. So the minimal difference between $t_m(d)$ and $t_e(d)$ has to be $(d-1)d/2d!$ and, again, the threshold $t_m(d)$ may be passed by $(d(d-1)-1)/2d!$.

$$t_e(d) - t_m(d) > \frac{2d(d-1)-1}{2d!} \tag{4.6}$$

The establishment of the matching is only possible, when the number of peers at any node is larger as $(d+1)^2$. This gives a minimal value for $t_m(d)$, depending on k:

$$t_m(d) > (d+1)^2 + k + \frac{d \cdot (d-1)}{2d!} \tag{4.7}$$

The last term takes into account, once more, that the result of the information aggregation algorithm has a delay of $(d-1)$ phases. Note that it has not explicitly to be guaranteed, that after an expand step, enough peers will be in each grid. This is included in Inequality 4.4, which guarantees that after an expansion, no reduction is necessary.

The Inequalities 4.1 to 4.7 should hold for all $d \geq 1$. For simplicity, we consider the coefficients of $t_r(d)$, $t_e(d)$ and $t_m(d)$ respectively to be integer constants. This is of course not necessary, since the thresholds are given in the average number of peers per node. Fractional constants could be more favorable in respect to the maximum number of peers per node. However, since the our goal is just to show the existence of the thresholds, rather then optimizing them, integer constants are used

From Inequality 4.5 we conclude $t_r(d+1) \geq (2+k)(d+1) + (2d^2-1)/2d!$. This is satisfied by

$$t_r(d) = (2+k)d + 4$$

for all $d \geq 1$.

To satisfy Inequality 4.1, we set

$$t_m(d) = (2+k)d^2 + (8+4k)d + k + 8$$

Further, satisfying Inequality 4.6, we set

$$t_e(d) = (2+k)d^2 + (6+2k)d + k + 10$$

It can easily be verified, that all other inequalities also hold with these thresholds. From the threshold for reducing, the minimal number of nodes in the system can be computed (for $d > 1$). The maximum discrepancy from the average kd, as well as the possibly removed tokens within the delay of the information aggregation algorithm, as well as the number of tokens per node the threshold may be passed have to be subtracted from t_r:

$$|v|_{min} = t_r(d) - kd - \frac{2d(d-1)-1}{2d!} = 2d + 4 - \frac{2d(d-1)-1}{2d!} > 2d$$

The maximum number of peers that can be at a node, is given by the threshold for expanding, plus the maximum discrepancy kd in the number of peers at a node from the average plus the number of peers that can join during the delay given by the token distribution algorithm plus the number of peers the threshold may be passed. Thus ending up with the maximum numbers of peers per node

$$|v|_{max} = t_e(d) + kd + \frac{2d(d-1)-1}{2d!} \leq (2+k)d^2 + (6+3k)d + k + 11$$

To conclude, a value for k has to be set. From chapter 2.3, we have $\phi \leq 4d + 3(J+L)$ as the maximum discrepancy in the number of peers per node between two arbitrary nodes in P_d. Since the joins and leaves are linearly superposed, we can set

$$k = 4 < \frac{\phi}{2d} = \frac{4d + 3(d/2 + d/2)}{2d} = 7d/2$$

Lemma 4.1. *With the thresholds $t_r(d) = (2+k)d + 2$, $t_m(d) = (2+k)d^2 + (6+2k)d + k + 6$, and $t_e(d) = (2+k)d^2 + (6+2k)d + k + 8$, it is guaranteed, that the number of peers in any node of the pancake system is always at least $2d+2$ and bounded by $\mathrm{O}(d^2)$.*

Proof. The lemma is proved with the above calculation. □

4.3 Properties of the Pancake System

This chapter will give a short overview over the properties of the pancake system. They are all based on the detailed discussion of the pancake system in the earlier chapters. Here we will present the most important results. We again assume that the pancake system is running in any order $d > 1$. The fact that our algorithm does never fail, is not made explicit here. However, this is the directly seen from the corresponding proofs in Chapter 3.

We first state that the pancake system on the end of each phase, is always repaired up to the churn of the current phase.

Theorem 4.2. *At the end of each phase, the pancake system is fully repaired up to $d/2$ joins and leaves.*

Proof. In each phase, the churn that happened to the pancake system before the current phase is repaired. This is done by $\mathcal{A}_{GRID}$ and $\mathcal{A}_{EDGE}$, as stated in the Lemmas 3.1 and 3.3. It is also shown for all further algorithms that may run during a phase, do not affect the grid. After $\mathcal{A}_{GRID}$ and $\mathcal{A}_{EDGE}$, the system is always fully repaired up to the at most $d/2$ joins and leaves that may occur during the current phase. For this, we observe that only the algorithms $\mathcal{A}_{EXP}$, $\mathcal{A}_{RED}$ and $\mathcal{A}_{TD}$ may change the connections of the grids. For these three algorithms, the desired is stated and proved in the Lemmas 3.10, 3.12 and 3.14 respectively. □

Corollary 4.3. *The pancake system P_d is always fully repaired up to at most d joins and leaves.*

Proof. From Theorem 4.2 and the following proof, we know that at the end, as well as after the run of $\mathcal{A}_{GRID}$, the system is always repaired up to $d/2$ joins and leaves. During the run of $\mathcal{A}_{GRID}$, there are at most d leaves and joins in the system: namely the up to $d/2$ not yet repaired joins and leaves of the last round, and the up to $d/2$ of the current round. □

The following corollary guarantees that there is always a core peer. Since data is stored in the core peers as will be explained later in Chapter 4.4, this will guarantee that no data is lost.

Corollary 4.4. *For each phase of the pancake system, there is always at least one core peer in each node, that does not leave during the whole phase.*

Proof. This follows from the (1) the above corollary and its prove, and (2) from the fact that a fully repaired grid has $d + 1$ core peers in each node. Further, in the expand step, the statement holds for the old core, until the new core is defined. □

Finally, we show that the out-degree of any peer is bounded by $\mathrm{O}(d)$. Moreover, all information that is needed by any peer to participate in the pancake system is bounded by $\mathrm{O}(d)$. This is optimal for an adversary that can join or remove $\Theta(d)$ peers in constant time.

Theorem 4.5. *As long as the pancake system P_d is run, the total information needed for any peer to preserve the topology is bounded by* $\mathrm{O}(n)$.

Proof. This is seen from the Lemmas 2.8, 3.2, 3.4, 3.6, 3.7, 3.11, 3.13, 3.15, 3.17 and 4.2 and the fact that after each phase, the old information is deleted up to the needed connections, namely row and column neighbors, additionally for core peers the $d - 1$ matching partner and finally, if $m > 0$, the up to d matching partners needed for preparing the expand step. □

4.4 Data and Routing

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.

As already mentioned in the introduction, the computing of the diameter of a pancake graph in an efficient way is an unsolved problem. However, the routing can be done in $\mathrm{O}(d)$, which approximates the shortest path up to a constant factor. This factor is shown to be less then 2. The following algorithm computes a path between two nodes $v = (\ell_1 \dots \ell_d)$ and $u = (\ell_{i_1} \dots \ell_{i_d})$, consisting of the nodes v_1 to v_{k-1}, where $k < 2d - 4$, yielding a diameter of the pancake graph of $2d - 3$.

Lemma 4.6. *Algorithm 4 computes a path between any two nodes in P_d. The path length is at most $2d - 3$.*

Algorithm 4 Path between two nodes

```
1:  $k := 0$
2:  $v_0 := v$
3:  $\ell_i$ is set to the $l^{th}$ literal of $v_0, \forall 1 \leq k \leq d$
4:  **for** $j := 0$ **to** $d - 3$ **do**
5:      **if** $\ell_{d-j} \neq \ell_{i_{d-j}}$ **then**
6:          find $l$, for which $\ell_l = \ell_{i_{d-j}}$
7:          $k := k + 1$
8:          $v_k = \rho_l(v_{k-1})$
9:          $k := k + 1$
10:         $v_k = \rho_{d-j}(v_{k-1}$
11:         $\ell_i$ is set to the $l^{th}$ literal of $v_k, \forall 1 \leq k \leq d$
12:     **end if**
13: **end for**
14: **if** $\ell_2 = \ell_{i_2}$ **then**
15:     $v_{k+1} = \rho_2(v_k)$
16: **end if**
```

Proof. The above algorithm computes an ordered set of nodes $v_0, \ldots, v_m$. We have $v_0 = v$. For all $v_j, 1 \leq j \leq k+1$, v_j v_j and v_{j-1} are neighbored since v_j is retrieved by a prefix-inversion of v_{j-1}. In step j of the for loop, the number $\ell_{i_{d-j}}$ is brought to position $d - j$. Beginning with $j = 0$, after the $d - 2$ iterations of the while loop, the latest computed node has the same numbers at positions 3 to d as v. This holds, since after iteration j of the while loop, all following prefix-inversion will be shorter as $d - j$. After the while loop, the last prefix inversion of length 2 is done if $v_k \neq u$. From the above we conclude, that the last computed node $v_m = u$. m can be at most $2d - 3$: Beginning with $k = 0$, the for-loop is passed $d - 2$ times, increasing k by at most 2, and in the end one further node may be added. m obviously corresponds to the path length. □

Lemma 4.7. *The routing between any two peers in our pancake system can be done in* $\mathrm{O}(d)$ *steps.*

Proof. Consider any two peers in the presented pancake system. First the message is sent to all row members by the sender, and then each peer that receives the message forwards it to the core peer in its column. It is guaranteed that one of the core peers will receive the message. Since we can not guarantee a complete column number during the complete sending, the core peers send the message to all members of the core. Then, the path as computed in Algorithm 4 is applied. Additionally, we insert for each node one step, forwarding the message to all core members. In the following round it then is sent to the core peers of the next node in the path. When the destination node is achieved, the message is sent into to the row of the destination peer. From there, the message can be sent to it's destination. After 3 rounds the message is broadcasted in the core of the senders node, then within the next $2(2d + 3)$ rounds the message is broadcasted in the core of the destination node, and finally, in the last 2 rounds the message is received, taking at most $4d + 11$ rounds. □

Note that the proof does not correspond to the best possible solution: Since during each phase, there exists a column number that is complete in all grids, the broadcasting through the core has only be done when a new phase begins.

In the following theorem, the observations above are summarized:

Theorem 4.8. *Our pancake P2P system guarantees node 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* 1 *living core peer and no data is lost. Moreover, it holds that* $d = \Theta(\frac{\log n}{\log \log n})$.

Proof. The node degree is a direct consequence of Theorem 4.5 and the network diameter in the above Lemma 4.7. The surviving core peer is stated in Corollary 4.4. That no data is lost, is directly derived from the surviving core peer and the fact that all data of a node is stored in each node peer. □

Chapter 5

Conclusions

We presented a P2P system which maintains desirable properties such as low peer degree and low network diameter against powerful, concurrent adversary which has complete visibility of the entire state of the system. We showed that the fault tolerance is asymptotically optimal as the robustness of any topology is trivially upper bounded by the peer degree.

We showed that our techniques allow to make a most intricate graph topology dynamic. We believe that majoring the pancake as a most intricate topology it may be possible to write a recipe for any P2P topology, by simply applying our basic components as ingredients.

The basic components we used were also used in [13]. We just applied them to the pancake graph, a topology, which is more complex and less intuitive. However, we developed various new ideas that may be used for other topologies too, two of them mentioned here: The node representation as a 2-dimensional grid allowed a node to consist of $\mathrm{O}(d^2)$ peers, still bounding the out-degree of the peers by $\mathrm{O}(d)$. Even if this is not done explicitly here, we state that the grid can be generalized to any dimension k. For each fixed k, this may increases the diameter of the grid by a constant. With this idea, the out-degree of a peer in a node, consisting of at most $\mathcal{O}(d^k)$ can be bounded by $\mathcal{O}(k)$. Since the diameter, and thus the number of rounds needed for the necessary algorithms will only be increased by a constant, this does not change the asymptotically results.

A further idea which was not used in [13], is that neighbored cores are only connected by a matching. With this idea, the out-degree of the hypercube system presented in [13], could be decreased from $log^2 d$ to $logd$, which is optimal since it corresponds to the out-degree of the hypercube. For this optimization, the number of rounds per phase in the hypercube system has just to be increased by 1.

Bibliography

[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)*, page 40.2, 2003.

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

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

[4] 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.

[5] D. Berman and M. Klamkin. A Reverse Card Shuffle. *SIAM Review*, 19:739–741, 1977.

[6] H. Dweighter. *American Mathematical Monthly*, 82, 1995.

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

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

[9] 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.

[10] M. Heydari and I. Sudborough. A Quadratic Lower Bound for Reverse Card Shuffle. In *Proc. 26th S.E. Conf. Combinatorics, Graph Theory, and Computing*, 1995.

[11] M. Heydari and I. Sudborough. On the Diameter of Pancake Networks. *J. Algorithms*, 25:67–94, 1997.

[12] 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.

[13] 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.

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

[15] 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.

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

[17] 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.

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

[19] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *Proc. USENIX Ann. Technical Conference*, 2004.

[20] 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.

[21] 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.

[22] B. Y. Zhao, L. Huang, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), 2004.

Appendix A

Acknowledgements

Finishing my diploma thesis in computer science, I look back to 4 months of hard working on a theoretical topic. My decision to choose for this topic was based on my affection for theoretical work. In contrast to other offers, this thesis seemed not narrowing to me, in the sense that it was open for my own ideas. I was also sure, that I would achieve interesting results.
Now, at the end of the diploma thesis, I can conclude that my expectations have been fulfilled: From the beginning, my ideas have been supported by my advisors and decided the following work. I am proud that with the help of my advisors, a conference paper on my work has been handed in. This gave me an additional insight to research in theoretical computer science.

I want to thank Fabian, Roger and Stefan for their support and various helpful ideas and feedback.