
# Telematics Institute

## Technical Report

# Performance of Lookup Operations in a Hypercube-based P2P Data Store: Theoretical Model and Performance Evaluation

Dietrich Fahrenholtz
Andreas Wombacher
Volker Turau

Dezember 2005

Report No. **TR-2005-12-01**


Hamburg University of Technology
Schwarzenbergstrasse 95, 21073 Hamburg, Germany
phone: +49 40 42878-3860; Fax:+49 40 42878-2581
email: telematik@tuhh.de
web: http://www.ti5.tu-harburg.de

# Performance of Lookup Operations in a Hypercube-based P2P Data Store: Theoretical Model and Performance Evaluation

Dietrich Fahrenholtz / Volker Turau
Telematics Institute
Hamburg University of Technology
{fahrenholtz,turau}@tu-harburg.de

Andreas Wombacher
Information Systems Group
Universiteit Twente
a.wombacher@utwente.nl

## Abstract

*One way for Peer-to-Peer data stores to achieve high data availability is to replicate their data. This is necessary to counter the effects of peer population dynamics also known as churn. A consequence of churn is that locating a data item may require a peer to resend search messages thus introducing additional communication. A formal model of this communication pertaining to data item lookups is introduced and evaluated using simulation in this paper. Results hold true for hypercube-based P2P data stores.*

## 1. Introduction

Peer-to-Peer (P2P) data stores are becoming more and more popular nowadays. They are used in several application domains besides the popular file sharing. For example, P2P systems maintain reputation or trust information of people transacting with each other [6] or support decentralized annotation of digital libraries [15]. Not every data store fits all requirements of these applications. Some demand storage of data with large sizes, others primary demand is a quick data insertion, update, and deletion, again others require strong data security. However, all of them attach importance to data availability when using a P2P data store, for people would not use these applications if their data were lost even unintentionally.

In this paper, the motivating application domain is decentralized maintenance of reputation information of people doing trade with each other. In particular, we consider an application scenario where a distributed trading system provides similar reputation information as, e.g., eBay [5]. But due to its distributed nature, no centralized party is available to maintain trading people's reputation information, nor is there a centralized infrastructure to store those data. As a consequence, a P2P data store built and supported by the people who trade with one another is needed that maintains their reputation information. A trader may want to perform at least two kinds of actions:

- He/She wants information about the person he/she is about to trade with. One can do this with a query for the reputation of the potential partner. This translates into a lookup in the P2P data store.

- He/She may want to report and store the experience obtained during a trade into the reputation system, which translates into an update of the P2P data store.

Characteristic for this application domain is the size of the data, which is comparatively small, while the required data availability must be high. However, the probability of a peer being online is rather small. Most likely, a peer will only be online for the time to select a trading partner, do business, and feed back his/her experiences. Therefore, the aim is to come up with a decentralized data store guaranteeing a configurable data availability without increasing the query complexity significantly. Both aspects - data availability and query response time - are crucial user acceptance factors for such a reputation system.

A high data availability P2P data store based on hypercubes and data replication within hypercube nodes has been introduced in [7]. Also the maintenance of the P2P data store has been presented there. In this paper, a theoretical model of the lookup part of a DHT layer operation is introduced and simulations with peer populations up to 10,000 peers are explored. It turns out that the complexity of lookups is independent of hypercube node size and the amount of data in the P2P data store, but depends on peer availability.

We summarize the basics of hypercubes and the high data availability extensions briefly in section 2. Section 3 provides a theoretical analysis of the approach, which is validated in section 4. Finally, a discussion of related work (section 7) and conclusions (section 8) follow.

### 1.1. Methodology

P2P systems are not a phenomenon that can be observed in the real world. However, the principle of self-organization, which is inherent in every P2P system, can be found in nature and adapted to their benefit. The peer population in P2P systems appears to be random and peers act very likely without interdependence. Probability theory and statistics are well-known and well-understood areas of mathematics. Their methods and results can help describe and grasp random events formally. When designing our P2P data store, we first try to model our observations of peer behavior formally and derive results from this model also in a formal way. Thus, by using mathematics, we make sure our results are valid and hold throughout our experiments. We are interested, though, in the behavior of a real-world application. Programming a version of our P2P data store that implements only minimum requirements is not a small project. Furthermore, we would require a huge amount of volunteers who would run the peer software on their computers interacting with one another. The many volunteers would be necessary to evaluate the scalability of our design. Those volunteers might be unwilling to test a P2P software that would inevitably contain flaws. That is why we set out to develop a simulator that adheres to our formal requirements and postpone the development of a working P2P data store to a later point in time. Since every program contains flaws, we use our formal model to verify the fitness of our simulator. By proceeding in that way, we can be sure there would be a great correspondence of what is happening while the simulator runs and what would happen if our P2P data store were to run in reality.

## 2. P2P Data Stores

Consider P2P networks that route packets to their destination using a greedy routing algorithm. Peers in these networks collaborate to form a structured overlay that helps in locating resources efficiently. To this end, they use an underlying network such as the Internet to connect to one another and a data structure called Distributed Hashtable (DHT) which is necessary to map a search key to a data item. Famous examples of these networks are Chord[17] and those whose topology is a $k$-ary tree like P-Grid[1] and Pastry[16], for instance.

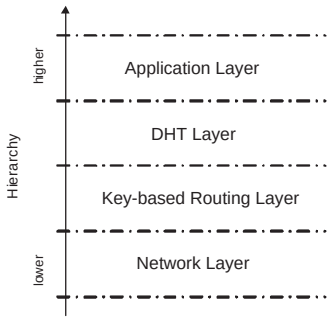


**Figure 1. Hierarchy of layers**

According to Leighton [10], a hypercube is a very powerful and well-understood network for parallel computation that can simulate the aforementioned networks efficiently. Therefore, we decided to let our P2P network be a hypercube. Since every P2P network implements a layered architecture, our P2P data store also follows the identification of fundamental layers done by Dabek et al. [4]. Figure 1 shows the abstractions. All basic operations of the DHT layer like *insert*, *update*, and *lookup* are dependent on successful routing of messages through the P2P network. In particular, DHT layer operations consist of two parts. The first is based on finding the data item that fits to a given search key or returning that there is no such data item, whereas the second does the actual data item manipulation. Thus, routing in hypercubes is a basic building block for the DHT layer above and for the analysis in this paper. But to make data actually available, our P2P network offers an additional, orthogonal high data availability concept.

### 2.1. Hypercube Networks

Every data item in a P2P network needs to be identified in a non-ambiguous way, which is accomplished by a key, $k \in \mathcal{K}$, where $\mathcal{K} = [0 \dots 2^\ell - 1]$ represents the key space. $k$ is derived from the data item by use of a collision-resistant hash function like SHA-1 [12], for instance. It also has the desired feature of distributing keys uniformly over $\mathcal{K}$. This guarantees a balanced utilization of the hypercube since every node in the hypercube is responsible for a unique range of keys, so every data item is maintained by exactly one node. By definition, a hypercube of dimension $d$ has $N = 2^d$ nodes, where the nodes are associated with an ID represented by a bit vector of size $d$. A node is connected with $d$ other nodes via bidirectional links. In particular, there is a connection between a node $u = u_1 \dots u_d$ and a node $v = v_1 \dots v_d$ if and only if their two IDs differ in exactly one bit.

Sending a message from node $u$ to node $v$ via such a link represents a **routing step**. This, however, requires routing tables maintained locally by each node. They store $d$ references to other nodes, one for each dimension. The aim is to greedily forward search messages to a node that is closer to the destination. A more detailed description is available in [7].

### 2.2. High Data Availability Concept

Our P2P data store extends the concept of a pure hypercube found in parallel computers. Here, a group of peers constitutes a hypercube node as opposed to a processor. Peers of such a data replication group[1] store the same data items in their memory. They work together to enable data redundancy which is necessary to achieve the required data availability since peers may leave the P2P data store at their own discretion making their data items unavailable. So a replication group is accountable for data availability in contrast to single peers in other P2P networks. Furthermore, the self-organization of our P2P data store guarantees sufficient replication group sizes without the need of periodical data item refreshment. In particular, if the number of peers increases or decreases greatly over a period of time, there are maintenance operations called *split* and *coalesce* which halve one or merge two hypercube nodes into one, respectively. These operations will be invoked in case the number of peers belonging to a node exceeds an upper threshold or falls below a lower threshold. Thus, high data availability can be guaranteed. Details on maintenance operations can be found in [7]. For the further understanding of our theoretical model, a description of the lookup algorithm extending the basic routing algorithm is required and provided next.

### 2.3. Lookup Operation

Recall that the overall objective of the lookup part of a DHT layer operation is to find the data item that fits to a given search key or to return that there is no such data item. Every routing step helps in attaining this objective by forwarding search messages to the node that is closer to the destination best in a greedy fashion. **Closeness** for our P2P network is defined as the size of an interval containing the search key: the smaller the interval the closer the destination.

Successfully routing a message requires implementing routing tables that contain references to other nodes. Every peer, $p_i$, has its own routing table $rt$, which consists of one entry per hypercube dimension. Every entry references a directly reachable node being responsible for an interval of data item keys. Such a reference in turn comprises of a list of peer references pointing to peers that belong to that node. In particular, a peer list consists of at least one peer reference but may contain as many peer references as there are peers belonging to that node. A peer reference is made up of IP address, port number, additional meta information such as measured online status of a remote peer, and packet round trip time.

In general, peers are considered to be online and connected to the P2P data store. These peers are called **active**, while those being offline or not connected are called **inactive**. Due to the dynamics of the peer population and the time span between two updates to routing table entries, a routing table may contain references to inactive peers. Having more than one peer reference in a peer list provides optional routes between two nodes, which support network cohesion in case of inactive peers. So the probability of a sudden network partition due to peer population dynamics decreases with the number of peers contained in the peer list.

The high data availability concept ensures data items stored in a replication group are available with high probability. But in order to be accessible, there must be at least one active peer per replication group serving requests and other peers of acquainted nodes must know of and can reach those remaining active peers. Since only active peers store data items, their accessibility also warrants data availability. So if we say a data item is available with high probability, we also imply it is accessible and there is an active peer serving requests for it. Algorithm 1 shows the message forwarding algorithm implementing a routing step, which is used by the lookup algorithm.

The aim of the algorithm 1 is to perform a routing step. This means, a message, $m$, which has been successfully sent to peer $p_i$ by the peer $p_h$ needs to be forwarded to any active peer of a node responsible for interval $interval$. To accomplish this, one of the first steps for $p_i$ is to get the current list of peer references belonging to the node that is responsible for interval $interval$ from $p_i$'s routing table (see line 3). Then, a peer, $p_x$, is randomly selected from $peerList$ (line 6) and $p_i$ attempts to send the message $m$ to $p_x$ (line 7). This is done by calling a network infrastructure primitive ($send$). Afterwards, peer $p_i$ sends an acknowledgment to peer $p_h$ (line 9) to signal message $m$ has been received and sent onwards successfully. Notice that acknowledgments will not be confirmed. Since the previous interaction with $p_h$ succeeded, $p_i$ is certain its answer will make it to $p_h$ with high probability. Peer

[1] In the following, we use replication group and hypercube node synonymously.

**Algorithm 1** performs a routing step

```
1: function FORWARD_MESSAGE(p_h, interval, m, C_min)
2:     varlocal peerList, counter, answer, p_x
3:     peerList ← rt[interval]
4:     counter ← 0
5:     repeat
6:         p_x ∈ peerList                    ▷ choose p_x randomly
7:         p_i.send(m, p_x)      ▷ send is a network primitive
8:         if counter = 0 then
9:             p_i.send('ACK', p_h)   ▷ send acknowledgement
10:        end if
11:        answer ← p_i.waitfor(p_x)
12:        counter ← counter + 1
13:    until answer = 'ACK' or counter ≥ C_min
14:    if answer = 'ACK' then
15:        return 'success'
16:    end if
17:    return 'failure'
18: end function
```

$p_i$ waits for an acknowledgment of peer $p_x$ blocking the execution (line 11) for a period of time. The purpose of acknowledgments is to ensure progress of a lookup operation and a repeat forward of the message $m$ to a different peer is not necessary. Calculating time-out values for acknowledgments to arrive is done outside this algorithm. If peer $p_x$ fails to answer within this time frame, peer $p_i$ considers $p_x$ to be inactive regardless whether the timeout was caused by not answering or due to a network problem like congestion, for example.[2] This minimizes but does not preclude the risk of a DHT layer operation to die due to a failure of intermediate peers that become inactive just after they have sent an acknowledgment. On the other hand, if an acknowledgment arrives late due to network problems, the search message might be duplicated. It is left to the DHT layer operation originating peer how to deal with duplicates in these rare cases. The whole process of selecting random peers and sending message $m$ to them (lines 6 through 13) is done as long as anyone of them answers, or a sufficient number, $C_{min}$, of peers from the peer list have been tried without success (line 13). Our notion of "sufficient" will be quantified in the next section. Finally, lines 14 through 17 serve to signal the calling DHT layer operation that either the routing step has finished successfully or not.

## 2.4. Upper bound on message forwarding attempts

We demand that there be at least one active peer per group present at any time that can serve requests for data items. According to Knežević et al. [9], the counter probability of all peers being inactive must be greater than the requested data availability guarantee, $a$, i.e. $Pr[A \geq 1] = 1 - {\pi_{inact}}^{C_{min}} \geq a$, where binomial distributed random variable $A$ corresponds to the number of available and accessible peers, $\pi_{inact}$ is the probability of a single peer being inactive, and $C_{min}$ is the minimum number of distinct peers belonging to each node. By fixing $a$ and a maximum allowable $\pi_{inact}$, we obtain $C_{min} \geq \frac{\ln(1-a)}{\ln(\pi_{inact})}$. An application using our P2P data store might require $a \geq .99$, a very high data availability guarantee, and allow $\pi_{inact} \leq .8$ the maximum, i.e., 80% of the peer population may leave at the most. So we compute $C_{min} \geq 20.638$ and this translates directly to there is at least one peer among 21 peers contacted by algorithm 1 that is still active with probability of 99%.

Although, as we will see in section 4, the number of peers of a group does not influence our results, it, however, gives us a necessary precondition. If the last active peer of a group leaves, remote peers will try indefinitely to contact formerly active ones of this group thus leading to an undesired, endless waiting of the lookup originating peer.

## 2.5. Bootstrapping of new peers

Briefly, we reiterate the steps of the join procedure described in [7]. Initially, a new peer that wants to join the P2P network has no knowledge about already existing peers, so it contacts either one of the peers known from a previous participation in the P2P network or some master bootstrap peer. The new peer gets a uniformly distributed random number, $k_{join} \in \mathcal{K}$, which serves as a join key. A node responsible for this key will be found by means of routing a `join_request` message to the destination node. The node coordinating peer then bootstraps the new peer by transferring group's data items, list of peers belonging to the node(aka. *neighbors*), and a complete routing table to it. After that, the new peer has become a full member of the replication group and is capable of routing messages and performing data storage layer operations like the others. Bootstrapping peers this way also pursues the goal of an even distribution of peers across all hypercube nodes, which serves load-balancing.

## 2.6. Maintaining routing tables

In this section we explain why maintaining routing tables is a challenging task. One of the prominent characteristics of P2P networks in general is that peers come and go as they like. This distinguishes P2P

[2] Calculating tight timeout values can be done according to the approach in [8]

networks from parallel computer architectures and distributed databases [13], for example. This means, routing table entries become out-dated over the course of time and need to be refreshed in order to eventually prevent a disconnection of nodes from one another. Freshness of routing tables can be achieved by updating entries in it frequently, which inevitably causes network traffic. Since bandwidth is a more valuable resource compared to CPU power and memory [3, 11] peers are reluctant to spend on maintenance, the less bandwidth consumption for maintenance the better.

## 3. Analysis of the routing protocol

In the following, a theoretical model of the mean communication overhead of the lookup part of DHT layer operations is introduced. To this end, we analyze the routing of messages through the hypercube and derive an equation that quantifies the mean path length over all lookups. Then we present an additional equation on the mean number of attempts peers need to make to forward a message to the next node thus completing a routing step. Finally, the two results are combined to quantify the mean lookup communication overhead for one lookup.

In order to carry out our analysis in a manageable way, we require our P2P data store to be in *steady state* or *equilibrium*, i.e., the number of peers in it remains constant over a period of time. If this were not so, the analysis would be much more complicate and its presentation would exceed available space. So we assume for the rest of the paper our P2P data store to be in steady state. Since peers join and leave at their own discretion, join and leave rates are equal and apply to the entire P2P data store. The amount of peers that leave a group has a direct impact on the perception of $\pi_{inact}$ pertaining to this group acquainted nodes experience. Thus the leave rate has an important influence on the communication overhead.

### 3.1. Mean path length

First of all, we need to define several properties. A **path**, $\tilde{n} = n_0, \ldots, n_j$, $j \leq d$, is a sequence of hypercube nodes of **path length** $j$ that results from the execution of the lookup part of a DHT layer operation starting at node $n_0$ and ending at node $n_j$, where node $n_{i+1}$ must be reachable from node $n_i$ and $0 \leq i < j$, thus representing a routing step. An active peer, $p_a$, belonging to node $n_i$ can **reach** an also active peer, $p_b$, belonging to node $n_{i+1}$ if $p_a$ can route a message directly to $p_b$ and $p_b$'s peer reference is stored in $p_a$'s routing table. If $p_a$ cannot find any active peer $p_b$ belonging to node $n_{i+1}$, then node $n_{i+1}$ is called **unreachable** from $p_a$. There may, however, be other peers belonging to node $n_i$ that can reach peers of node $n_{i+1}$. This phenomenon is called an overlay network anomaly. If the P2P network consists of $2^{d-1} < v \leq 2^d$ replication groups, we let $v = 2^d$ which serves for calculating conservative mean path length values.

**Theorem 1.** *Let $X$ be the event of finding an arbitrary but fixed key, $k$, starting from any node, $n_0$. The probability of reaching the destination node responsible for $k$ in exactly $s$ steps with $0 \leq s \leq d$, is*

$$Pr[X = s] = \frac{\binom{d}{s}}{v} \tag{1}$$

*where $v$ is the number of nodes in our P2P network, which is in steady state. The expected number of steps is*

$$E[X] = \frac{d}{2} \tag{2}$$

*for any DHT layer operation.*

*Proof.* First of all, observe the probability of node $n_0$ to be responsible for key $k$ in step 0 is $\frac{1}{v}$. With probability $1 - \frac{1}{v}$, key $k$ needs to be routed onwards. Every peer of node $n_0$ has $g \geq 1$ pointers to acquainted nodes. Thus they have the choice which of the $g$ reachable intervals is most suitable. So the probability of node $n_1$ to to be responsible for key $k$ increases to $\frac{g}{v}$.

We show the number of distinct hypercube nodes that can be reached in step $s$ from all hypercube nodes of the previous step equals $\binom{d}{s}$. Our proof is by induction on the number of steps.

*Initial step:* $\binom{d}{0} = 1$. If $k$ is element of the interval $n_0$ is responsible for, no further routing is required. $\binom{d}{1} = d$. If $k$ is element of one of the intervals of hypercube nodes one step further, then by definition of the hypercube, there can only be $d$ potential nodes. Now we assume $\binom{d}{s-1}$ holds for all $s - 1 > 0$. Also keep in mind that $\sum_{i=0}^{s-1} \binom{d}{i}$ distinct hypercube nodes have been checked for responsibility in the previous $s - 1$ steps.

*Inductive step:* Our assumption holds when transitioning from step $s-1$ to $s$ if we can show $\binom{d}{s} = z \cdot \binom{d}{s-1}$ by determining $z$. Whenever a message proceeds one step further, it reaches sub-cubes of dimension $d-s+1$. This feature of the lookup algorithm was proven earlier in [7]. So in step $s$, $(d - s + 1) \cdot \binom{d}{s-1}$ potential hypercube nodes can be reached by $\binom{d}{s-1}$ nodes. Observe, however, the set of potential nodes contains duplicates. This is because any hypercube node from this set can be reached via $s$ different paths. To see this, recall that any hypercube node can be reached by any other node

in $s-1$ steps, if the two nodes differ in exactly $s-1$ bits. That means there are $(s-1)!$ ways to route a message between these two nodes. By the same argument, there are $s!$ ways overall when performing step $s$. But our algorithm makes sure, any node will be reached in precisely one way thus eliminating duplicate hypercube nodes. But when there is exactly one path, we divide the number of possible nodes in step $s$ by $s$ yielding a total of $z = \frac{d-s+1}{s}$ nodes. Multiply this $z$ with $\binom{d}{s-1}$ and the inductive step follows.
The expected number of steps for a DHT layer operation follows directly from the definition of the mean.

$$E[X] = \sum_{i=0}^{\infty} i \cdot Pr[X=i] = \sum_{i=0}^{d} \frac{i\binom{d}{i}}{v} = \frac{1}{v}\sum_{i=1}^{d} \frac{id}{i}\binom{d-1}{i-1}$$
$$= \frac{d}{v}\sum_{i=0}^{d-1}\binom{d-1}{i} = \frac{d}{v}2^{d-1} = \frac{d}{2}$$

where the last equality follows letting $v = 2^d$ thus completing the proof. □

### 3.2. Mean number of attempts

In what follows, $p_{x_j}$ denotes the $j^{th}$ peer selected from *peerList* by $p_i$ in step 6 of algorithm 1 ($j \geq 1$). If $p_{x_j}$ turns out to be inactive or unreachable from $p_i$, $p_{x_{j+1}}$ is selected uniformly at random from *peerList* and $p_i$ retries to send the message. Notice that $p_{x_j}$ is not removed from *peerList*. The number of attempts is proportional to the number of inactive or unreachable peers. To see this, we show the following theorem.

**Theorem 2.** *Let $\pi_{inact}$ be the probability of any peer being inactive and let $\pi_{fail}$ be the probability of a message getting lost while in transit. Let $Z$ be a random variable corresponding to the number of attempts a peer, $p_i$, needs to make to forward a message to a peer, $p_{x_z}$, of the next routing step. The probability of $p_i$ to try exactly $(p_{x_1}, ..., p_{x_{z-1}}, p_{x_z}), z \geq 1$ peers, where $p_{x_z}$ is an active and reachable peer and the other $p_{x_j}, 1 \leq j \leq z-1$ are not is*

$$Pr[Z=z] = q \cdot (1-q)^{z-1} \tag{3}$$

*and the expected number of attempts is*

$$E[Z] = \frac{1}{q} \tag{4}$$

*where $q = (1-\pi_{inact})(1-\pi_{fail})$.*

*Proof.* Clearly, the probability of any peer, $p_x$, being active $(1-\pi_{inact})$ is independent of the probability of a message reaching $p_x$ $(1-\pi_{fail})$. Thus the conditional probability of both is $q$. $Z$ is a geometric random variable. To see this, note all message forwarding attempts are independent of one another and the probability of all $z-1$ being vain attempts is $(1-q)^{z-1}$. If the $z^{th}$ attempt is the first successful one, this occurs with probability $q$. The conditional probability of encountering exactly $z-1$ vain and 1 successful attempts follows by multiplying both probabilities yielding equation 3.
The expected number of attempts is calculated according to the general definition of the mean. $E[Z] = \sum_{i=0}^{\infty} i \cdot Pr[Z=i] = \sum_{i=0}^{\infty} iq(1-q)^{i-1} = \frac{1}{q}$ The last equality follows when substituting $x$ for $q$. □

### 3.3. Mean communication overhead

Now we are able to combine mean path lengths generated by a DHT layer operation and mean number of attempts occurring at each routing step. Clearly, forwarding peers have to make $E[Z]$ attempts on average to complete a routing step. $E[X]$ steps on average are necessary for a search message to arrive at its destination thus yielding

$$E[X] \times E[Z] = \frac{d}{2q} = \frac{d}{2(1-\pi_{inact})(1-\pi_{fail})} \tag{5}$$

This formula is especially beneficial for peers that want to estimate the time a DHT layer operation takes to produce a response on the existence of a data item. Multiply equation 5 with an upper bound on the mean latency on network links and peers have a good figure on how long they have to wait in the average case.

## 4. Experimental validation

In this section, we describe our simulator and then the experiments we conducted to show that the prototypical implementation of our P2P data store behaves as predicted. To this end, we simulated different P2P data store sizes (number of hypercube nodes) in combination with different values of $q$ to see if a sufficient variety of simulation results fit to the theory.

### 4.1. Features of our simulator

We developed an activity-based simulator that allows for concurrent, non-deterministic peer interactions. We found this approach more suitable than a pure discrete event simulation, a view that is also supported by authors of [18]. Our simulator is written in Java and thus runs on many platforms. However, resource consumption is considerable. Simulating 10,000 peers requires more than 4 GB of main memory. This

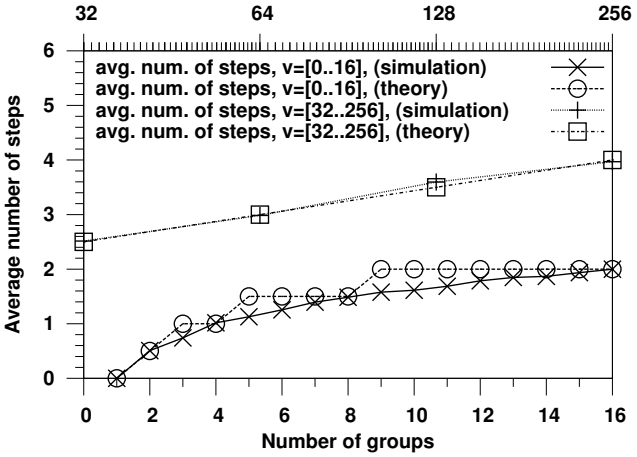


**Figure 2. Number of routing steps for lookups. ($\pi_{inact} = 0$, $\pi_{fail} = 0$)**

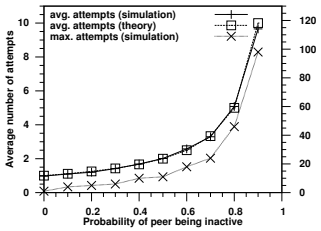


**Figure 3. Number of attempts lookups need varying $\pi_{inact}$. ($\pi_{fail} = 0$)**

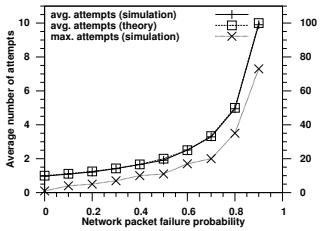


**Figure 4. Number of attempts lookups need varying $\pi_{fail}$. ($\pi_{inact} = 0$)**

is because every peer runs in its own thread. Experimenters can set many parameters. For example, there is a directive that configures which P2P network is to be simulated. Replication group parameters such as group size in case of our P2P data store can be also be set. Currently, the simulator only supports our P2P data store. We plan to simulate other P2P networks with it to make simulation results between them more comparable. Furthermore, parameters pertaining to the network such as packet failure percentages and latencies can be specified. Finally, one is able to configure the type, the frequency in percent, and the total number of data store layer operations that are to occur in a simulation phase. Additionally, the mean interarrival time between two consecutive data store layer operations and the configuration of how statistics should be gathered can be tuned.

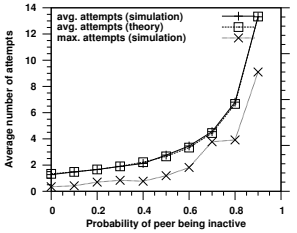


**Figure 5. Number of attempts with both failure models varying $\pi_{inact}$. ($\pi_{fail} = .25$)**

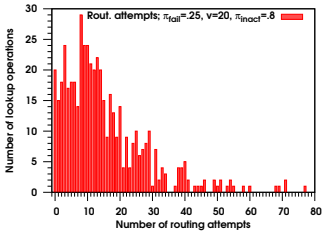


**Figure 6. Histogram of attempts needed by 495 lookup operations.**

### 4.2. Simulation setup

The core of each simulation experiment comprised of a number of lookup operations, where each of them starts from a randomly chosen active peer to find an also random search key and records results on the number of attempts and routing steps a lookup operation needs in order to be successful. Notice, a lookup operation guarantees to always find a data item if it is stored in the P2P data store. Furthermore, a simulation experiment is divided into two phases, where the first phase sets up the P2P data store. For all of our experiments, we configured $j$ peers to join the initially empty P2P data store at first to grow it to $v$ nodes. Recall, whenever a node contains more active peers than is permitted by a configurable upper bound, the node splits thus leading to P2P network growth. Subsequently a fixed number of insertions of distinct data items ensue. After that, $f$ randomly chosen peers leave to shrink our P2P data store to a target peer population of $N = j - f = \pi_{inact} \cdot j$ completing the first phase. So the mean number of peers per group amounts to $\frac{j-f}{v}$. Acting in this way is necessary because simply having $j - f$ peers join the P2P data store would not lead to the desired probability of a peer being inactive, $\pi_{inact}$. After setting up the P2P data store in this way, the number of peers and thus the structure of the P2P network remain constant.

The second phase serves the measuring of actions generated by lookups. In particular, the overall measurement is divided into several units, where each unit comprises of a predefined number of lookup operations. Units have been introduced to reduce the data volume to be handled by the peers. In this evaluation, every experiment has 10 units, each of them performs 500 lookups that write simulation results to a statistics file. All activity starts occur with a Poisson distributed in-

terarrival rate of 50 msecs. We fixed the lower bound of peers, $C_{min}$, belonging to a replication group to 50. This number is reasonable because it allows us to simulate hypercubes with dimensions up to 8 without exceeding our memory limit of 4 GB. Even if 90% of the peers leave, at least one peer per node remains active in the majority of cases. Moreover, we found out varying this bound did not have an impact on the experiments.

**4.2.1. Results on mean path length.** In figure 2 various mean path lengths are shown for P2P data store sizes from 0 to 256 nodes. We fixed the probabilities $\pi_{inact}$ and $\pi_{fail}$ to 0 as we found varying these parameters did not have an impact on the outcomes. For the two upper lines, the upper x-axis is applicable using a logarithmic scale. Results from theory (see Equation 2 in section 3.1) and practice match very well. The relative error is 2.74% the maximum. For the lower two lines, notice, simulation results are better than or at least equal to results from theory because we defined $E[X]$ as an upper bound on the average number of routing steps in section 3.1. However, the error between simulation and theory gets larger as the number of dimensions, $d$, increases due to the larger difference between actual number of nodes $v \geq 2^{d-1} + 1$ and the maximum number of nodes $v = 2^d$.

**4.2.2. Results on mean number of attempts.** Figure 3 presents results from an experiment in which we fixed the probability of network failures, $\pi_{fail} = 0$ and the number of nodes, $v = 16$, and varied the probability of an inactive peer, $\pi_{inact}$, from 0 to .9 in .1 steps. The average number of attempts for a peer to forward a message to a remote peer matches closely the calculated values from theory (see Equation 4 in section 3.2). The relative error is 3.4% the maximum. Varying the number of nodes $v$ did not make any difference as was expected that's why we omit these figures here. Each cross on the lower line resembles the maximum number of attempts a peer has to make to forward a message, that is, the number of applied routing steps. Note the y-axis on the right is applicable to this line. Actually, the chances for a peer to re-try this often are very low. We measured 1 occurrence in 5000 lookup operations. Lines shown in figure 4 result from a converse experiment, where we varied the probability $\pi_{fail}$ from 0 to .9 and fixed the probability $\pi_{inact} = 0$ and the number of nodes $v = 16$. Here, too, the average number of attempts matches closely corresponding values from theory (see Equation 4 in section 3.2). The relative error is less or equal 1%. Maximum number of attempts are again shown as crosses on the lower line using the right y-axis.

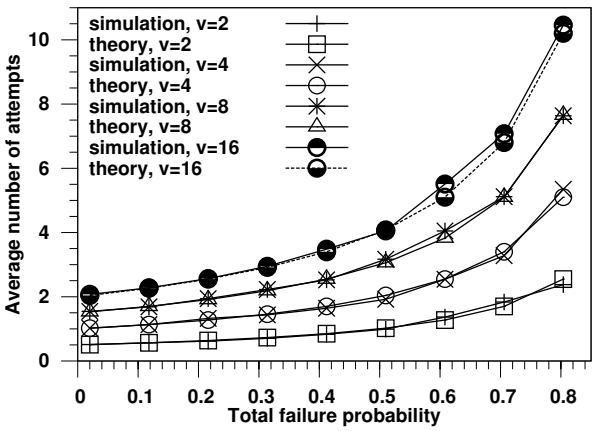


**Figure 7. Total number of attempts over network sizes 2 to 16 groups.**

In the third simulation, figure 5, we fixed the probability $\pi_{fail} = .25$ and the number of nodes $v = 20$ and varied the probability $\pi_{inact}$ again from 0 to .9 to show that these simulation results conform to the theory when messages can get lost either due to peer or network failure or both. The calculated (see equation 4 in section 3.2) and simulated average number of attempts match very well. The relative error is less or equal 2%.

The objective of the forth experiment was to get more information on the distribution of the total number of routing attempts a successful lookup operation needs. We set up a reference phase containing 500 lookup operations and let probability $\pi_{fail} = .25$, number of nodes $v = 20$, and probability $\pi_{inact} = .8$. Figure 6 shows the frequency distribution. 5 lookup operations failed because the forwarding peer could not find any active peer in its routing table for the next hop. This happened due to the high $\pi_{inact}$. More than half of the 495 lookups succeed using up to 12 and more than 90% need less than 30 routing attempts to find a data item. The remaining 10% need 31 to 70 attempts.

**4.2.3. Results on mean communication overhead.** In the next eight experiments, we were interested in whether the total number of attempts summed over the lookup part of a DHT layer operation followed the theoretical values calculated with equation 5 in section 3.3. So we varied the number of nodes $v$ from 2 to 256 and probability $\pi_{inact}$ from 0 to .8 in .1 steps. We fixed probability $\pi_{fail}$ to a more realistic value of .02, a very conservative estimate IP service providers surpass easily in general. Other simulation parameters are set up as mentioned above. To obtain accurate re-

sults, we calculated the average over all results from 10 simulation phases per value of nodes $v$, where a phase again comprised of 500 lookup operations. Figure 7 shows results both from our theoretical model and the simulation experiments. As we expected, values from theory match nicely with those from the simulation. The relative error is between 0.1% and 6.78% and thus acceptable. We refrained from displaying extreme values of number of attempts in order to keep the figure clear. We also do not display a diagram for results of P2P data store sizes greater that 16 nodes because it would not show any additional, not yet known information concerning lookup operation performance. This shows, equation 5 gives a good estimation on the total number of attempts a lookup operation needs on average. It also shows that increasing the failure probabilities linearly increases the number of attempts exponentially, so low failure probabilities are highly desirable. Note, the estimation holds even in case of high peer population dynamics as long as $\pi_{inact}$ and $\pi_{fail}$ remain constant throughout the execution time of a lookup operation.

## 5. Analysis of a routing protocol modification

The original routing protocol deemed peers inactive if no answer arrived within a previously specified timeout period. Such peers remained in peer lists causing their potential re-selection in subsequent rounds thus increasing the total time for a lookup operation unnecessarily. In this section we remedy this inefficiency by excluding inactive peers from further consideration. To this end, we develop a new equation for mean number of attempts of a routing step. The mean path length of a lookup stays unaffected of this modification. Also for this analysis, the network must be in steady state.

To modify algorithm 1 for our purposes, we remove variable *counter* because it is unnecessary now and insert a statement that removes peer $p_x$ from variable *peerList* after the message has been sent to it. In the new algorithm, line 13 is modified to check whether the number of peers contained in *peerList* has reached 0 or $p_x$'s answer arrived in time. If none of the two cases is true, then the selection procedure starts over again. The remaining parts of algorithm 1 do not change.

### 5.1. Mean number of attempts (mod. algorithm)

The setup is the same as in section 3.2 but this time, after testing $p_{x_j}$ for being responsive, we remove it from *peerList* regardless of whether its answer arrived in time or not. For the following theorem, the meaning of the variables of theorem 2 remain the same, however the equation for the probability changes. We would also like to give an equation in closed form for the mean number of attempts but it is prohibitively long and unwieldy, so we refrained from doing this. Instead, we calculate the expected value numerically for our experiments.

**Theorem 3.** *Let all variables be the same as in theorem 2. Let $|Pl|$ be the mean size in peers of a peer list, $q\,|Pl|$ the mean number of active peers in $Pl$, and $q = (1-\pi_{inact})(1-\pi_{fail})$ the probability of a message to reach an active peer that answers in time. Further let $B = (1-q)\,|Pl|$ an auxiliary variable. The probability of $p_i$ to try exactly $z \geq 1$ peers, where $p_{x_z}$ is an active and reachable peer and the other $p_{x_j}, 1 \leq j \leq z-1$ are not is*

$$Pr[Z' = z] = \frac{q\,|Pl|}{|Pl| - z + 1} \cdot \left( \frac{B!(|Pl| - z + 1)!}{|Pl|!(B - z + 1)!} \right)^{1-\delta_{1,z}} \tag{6}$$

*where $\delta_{i,j}$ is the* KRONECKER *symbol which is 1 if $i = j$ and 0 otherwise. The expected number of attempts can be calculated by*

$$E[Z'] = \sum_{i=1}^{|Pl|-1} i \cdot Pr[Z' = i]$$

*Proof.* Like $Z$, $Z'$ is also a geometric variable. The reasoning is the same as in the proof of theorem 2. So the probability of event $Z'$ is of the form $Pr[Z' = z] = \pi(z) \prod_{l=1}^{z-1} (1 - \pi(l))$, where $\pi(z)$ denotes the probability of step $z$ being a successful routing step, whereas $\prod_{l=1}^{z-1} (1 - \pi(z))$ is the probability that the previous $z-1$ steps have been failures. $\pi(z)$ must depend on $z$ because the probability of selecting an active and reachable peer from $Pl$ becomes larger as $z$ increases since peers that do not answer within a timeout period are removed from $Pl$. So the probability of performing a successful routing step is

$$\pi(z) = \frac{q\,|Pl|}{|Pl| - z + 1}\,, \qquad 1 \leq z \leq |Pl|\,. \tag{7}$$

Likewise the probability of performing $z-1$ unsuccess-

ful routing steps is

$$\prod_{l=1}^{z-1}(1-\pi(l)) = \prod_{l=1}^{z-1}\left(\frac{|Pl|-l+1-q\,|Pl|}{|Pl|-l+1}\right)$$
$$=\frac{|Pl|-q\,|Pl|}{|Pl|}\cdot\frac{|Pl|-1-q\,|Pl|}{|Pl|-1}\cdots\frac{|Pl|-z+2-q\,|Pl|}{|Pl|-z+2}$$
$$=\frac{B}{|Pl|}\cdot\frac{B-1}{|Pl|-1}\cdots\frac{B-z+2}{|Pl|-z+2}$$
$$=\frac{B!(|Pl|-z+1)!}{|Pl|!(B-z+1)!}\,,\qquad 1<z\leq|Pl|\,. \tag{8}$$

If $z = 1$, then there is no previous routing step and $Pr[Z' = z]$ must equal $\pi(z)$. We can accomplish this by raising equation 8 to $1-\delta_{1,z}$, which becomes 0 only if $z = 1$. Since all routing attempts are independent of one another, multiplication of equations 7 and 8 yields the desired result.
$E[Z']$ is the definition of the mean. □

### 5.2. Mean communication overhead (mod. algorithm)

Multiplying values for mean path length and mean number of attempts yields

$$E[X]\cdot E[Z'] = \frac{d}{2}\cdot\sum_{i=1}^{|Pl|-1} i\cdot Pr[Z'=i] \tag{9}$$

which is the mean communication overhead for the modified algorithm. Having equation 9 in closed form would be desirable, however, it would be very unwieldy and lack conciseness. This unfortunately also blocks an asymptotic analysis comparing equation 5 with 9.

## 6. Experimental validation of the modified algorithm

The experimental setup remained the same as in section 4 except for the original peer selection algorithm which is replaced by the modified one. For the new experiments, we again simulated different P2P data store sizes with varying values of $q$. Thus we could validate the new theorem.

**Results on mean communication overhead (mod. algorithm).** First of all, we wanted to know if values calculated with equation 9 match with simulation results. So we set up experiments like in section 4.2.3. We varied the number of nodes, $v$, from 2 to 256 and probability $\pi_{inact}$ from 0 to .8 in .1 steps. Probability $\pi_{fail}$ was fixed at .02. Furthermore, we calculated

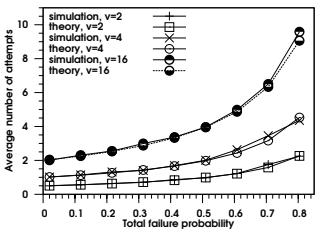


**Figure 8. Matching values from theory with ones from simulation, $v \in \{2, 4, 16\}$**

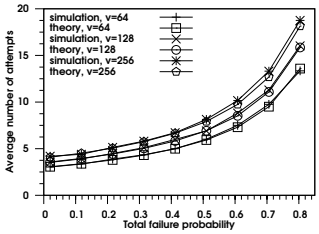


**Figure 9. Matching values from theory with ones from simulation, $v \in \{64, 128, 256\}$**

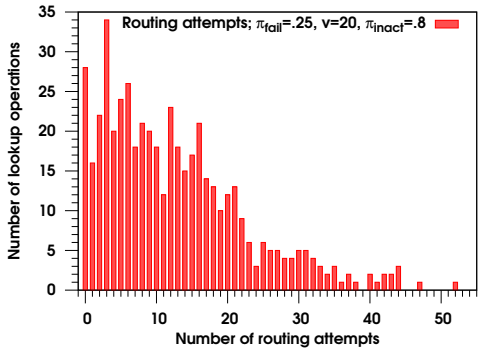


**Figure 10. Histogram of routing attempts needed by 495 lookup operations using the mod. algorithm**

the average over all results from 10 simulation phases per value of nodes v and had 500 lookup operations per phase. Figures 8 and 9 show that both theoretical values and simulation results match with a maximum relative error of less than 5.8%. To keep these figures clear, we did not show curves for 8 and 32 nodes. The choice is arbitrary and we could have taken any of the 8 curves. However the statement of the figures remains unaffected.

We were also interested in how many lookup operations need a particular number of attempts to route a message to its destination. This time we expect the number of attempts to be significantly less than those generated by the original algorithm. We again used a reference phase comprising of 500 lookup operations of which 495 succeeded. 5 lookup operations failed because a forwarding peer could not find any active remote peer that was responsible for the search key and answered in time. The probability of a network failure was $\pi_{fail} = .25$, the number of nodes was

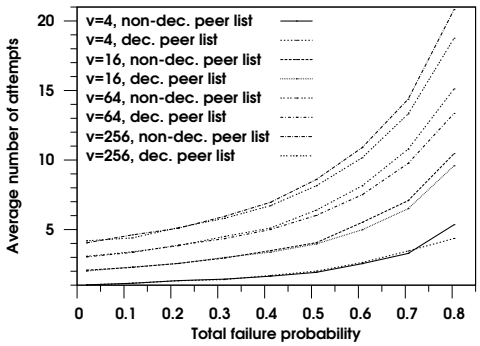


**Figure 11. Comparison of the performance of the first and second algorithm. The second disregards peers found to be inactive.**

$v = 20$, and the probability of a peer being inactive was $\pi_{inact} = .8$. Figure 10 shows the distribution of the number of lookup operations over the routing attempts. Comparing figure 6 with figure 10, one can see that in both figures the majority of lookup operations need fewer than 15 routing attempts overall. However looking at the maximum number of attempts, figure 6 shows a much longer tail. So the modified algorithm is effective in eliminating outliers. We now discuss if it actually decreases the number of attempts.

Figure 11 shows the direct comparison between the performance of the original algorithm that does not decrease the size of peer lists and the second algorithm that does. There is no doubt the modified algorithm needs fewer attempts to accomplish a routing step if the total failure probability is high. However, this effect is less noticeable or even not existing if the number of nodes is small ($2 \leq v \leq 16$) and the total failure probability is also small ($0 \leq q \leq 0.3$). This means the modified algorithm is preferable for P2P networks where either the network size is large ($\gg 10000$) or the failure probability is quite high ($> 0.4\%$).

## 7. Related Work

Distributed database systems are a widely researched field [13]. They are capable to distribute inserted data to available servers in the system so that high data availability on different physical database levels, e.g. by using erasure coding or replication, can be attained. They contrast with high availability P2P data store in that they need a centralized part controlling transactions, for instance.

Erasure codes have been applied to high available P2P data stores [2]. In particular, this method gives better file availability compared to other approaches, in case of very large file sizes and a high probability of peers being inactive. Neither of the above mentioned applies to the scenario investigated in this paper. Replication has been applied to P2P systems e.g. in [9, 14]. However, these approaches require the Distributed Hashtable to manage replication, whereas in our P2P data store hypercube nodes are responsible for this. Authors of [11] compare the performance of some DHTs under varying $\pi_{inact}$. The effects of peer population dynamics (aka. churn) show the same consequences as we found in our investigation. We, however, provide a formal model to explain our results.

## 8. Conclusion

In this paper, we briefly described our P2P data store. Its routing topology is a hypercube and it ensures high data availability. One of its prominent features is that hypercube nodes are data replication groups which guarantee any datum once inserted is available at every point in time with high probability. Other popular P2P networks do not have this feature of automatic data replication within nodes. Based on this approach, we introduced a theoretical model about the average total number of routing steps for the lookup part of DHT layer operations and validated the model by extensive simulations. As it turned out, theoretical values matched simulation results with a relative error of less then 6.8% the maximum. We then improved the selection of peers from peer lists in every routing step. To this end, we modified the original selection algorithm slightly. The resulting algorithm removes peers from consideration that proved to be inactive or did not respond within a timeout period. Here again the new theory that fits to this algorithm matches with simulation results. We expected the modified algorithm to perform better than the original one in terms of using fewer routing attempts. Comparison of both showed that the modified algorithm out-performs the original one when total failure probability is greater than 40% and the number of nodes in the P2P network is greater than 64. Although we simulated P2P network sizes up to and including 256 nodes, we are positive our theory holds true for larger sizes.

Future work will draw on this model to resolve the conflict between frequent routing table updates to achieve best lookup performance and their bandwidth consumption. Although our modified peer selection algorithm performs well, it still exhibits an exponential increase in the number of attempts if the total failure probability raises. Finding a sub-exponential algorithm is a challenging task, but it would aid signifi-

cantly in reducing routing table updates. Thus we plan to investigate this option in the future. Furthermore, our theoretical model will serve as a basis for the estimation of costs pertaining to the self-organization of replication groups.

## References


[1] K. Aberer: "P-Grid: A self-organizing access structure for P2P information systems", in *Proc. of the 6th Intl. Conf. on Cooperative Information Systems*, LNCS 2172, 2001, pp. 179 –194.

[2] R. Bhagwan, D. Moore, et al.: "Replication strategies for highly available peer-to-peer storage", in *Proc. of the Intl. Workshop on Future Directions in Distributed Computing*, LNCS 2584, 2002, pp. 153–158.

[3] Ch. Blake, R. Rodriguez: "High availability, scalable storage, dynamic peer networks: Pick two", in *Proc. of the 9th Workshop on Hot Topics in Operating Systems*, USENIX Assoc., 2003.

[4] F. Dabek, B. Y. Zhao, et al.: "Towards a common API for structured Peer-to-Peer overlays", in *Proc. of the 2nd Intl. Workshop on Peer-to-Peer Systems*, LNCS 2735, 2003, pp. 33–44.

[5] eBay: ebay.com, 1995, [online: http://www.ebay.com]

[6] D. Fahrenholtz and W. Lamersdorf: "Transactional security for a distributed reputation management system", in *Proc. of the 3rd Intl. Conf. on Electronic Commerce and Web Technologies*, 2002, pp. 214–224.

[7] D. Fahrenholtz and V. Turau: "A tree-based DHT approach to scalable weakly consistent data management", in *Proc. of the 1st Intl. Workshop on P2P Data Management, Security and Trust*, 2004, pp. 991–998.

[8] P. Karn and C. Partridge: "Improving round-trip time estimates in reliable transport protocols", *ACM Trans. on Computer Systems*, 9(4), 1991, pp. 364–373.

[9] P. Knesevic, A. Wombacher, and T. Risse: "Enabling high data availability in a DHT", in *Proc. of the 2nd Intl. Workshop on Grid and P2P Computing Impacts on Large Scale Heterogeneous Distributed Database Systems*, 2005.

[10] F. T. Leighton: *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Pubs., San Mateo, USA, 1992.

[11] J. Li, J. Stribling, et al.: "Comparing the performance of distributed hash tables under churn", in *Proc. of the 3rd Intl. Workshop on Peer-to-Peer Systems*, LNCS 3279, 2004, pp. 87–99.

[12] NIST: Us secure hash algorithm 1 (SHA1), 1994.

[13] M. T. Özsu and P. Valduriez: *Principles of Distributed Database Systems.* Prentice Hall, 1999.

[14] S. Rhea, C. Wells, et al.: "Maintenance-free global data storage", *IEEE Internet Computing*, 5 (5), 2001.

[15] T. Risse and P. Knežević: "A self-organizing data store for large scale distributed infrastructures", in *Proc. of the Intl. Workshop on Self-Managing Database Systems*, 2005, pp. 9–16.

[16] A. Rowstron and P. Druschel: "Pastry: Scalable, decentralized object location and routing for large-scale Peer-to-Peer systems", in *Proc. of the Intl. Conf. on Distributed Systems Platforms*, 2001, pp. 329–350.

[17] I. Stoica, R. Morris, et al.: "Chord: A scalable peer-to-peer lookup service for internet applications", in *Proc. of the ACM Special Intererst Group on Data Communications*, 2001, pp. 149–160.

[18] R. von Behren, J. Condit, et al.: "Why events are a bad idea (for high-concurrency servers)", in *Proc. of the 9th Workshop on Hot Topics in Operating Systems*, 2003, pp. 19–24.