Analyzing and Modifying Chord's Stabilization Algorithm to Handle High Churn Rates

Gerald Kunzmann, Andreas Binzenhöfer and Robert Henjes

***Abstract*—Peer-to-Peer (P2P) networks offer reliable and efficient services in different application scenarios. In particular, structured P2P protocols (like Chord [1]) have to handle changes in the overlay topology fast and with as little signaling overhead as possible. This paper analyzes the ability of the Chord protocol to keep the network structure up to date, even in environments with high churn rates, i.e. nodes joining and leaving the network frequently. High churn rates occur, e.g., in mobile environments, where participants have to deal with the limited resources of their mobile devices, such as short battery lifetimes or high communication costs. In this paper, we analyze different design parameters and their influence on the stability of Chord-based network structures. We also present several modifications to the basic Chord stabilization scheme, resulting in a much more stable overlay topology.**

***Index Terms*—Chord, Churn, Stabilization, Structured P2P**

I. Introduction

Most currently applied P2P networks are based on reactive routing. That is, if a node searches for any content, its lookup request is flooded through the network. Each node that receives a lookup request checks if it can answer the lookup. Otherwise it forwards the request to one, many or all nodes in its node cache. To prevent routing messages from being forwarded infinitely, a TTL counter is decreased with every hop and the request is discarded if the counter value is zero. Unstructured P2P networks have two main disadvantages. First, flooding the networks leads to high traffic load. Hierarchical architectures, such as Gnutella v0.6 [2], that introduce Superpeers, can significantly reduce the required bandwidth. The second drawback is that it is not possible, in large networks, to flood the request to all participants in the network. The small world paradigm (discussed in [3] Chapter 2.2) predicts that most lookups can be resolved in a median of five to seven hops. This is in particular true for popular content. However, it is less probable that unique or rare content can be found by flooding only a part of the network.

To overcome these disadvantages, a new generation of P2P networks, the structured P2P protocols have been developed and became one of the most active P2P research areas. Structured P2P networks avoid flooding by building a well defined overlay structure among the participating nodes. In this context, each node is assigned to a unique ID, which is for example generated by hashing the node's IP address into a *d*-dimensional ID space. Additionally, each node keeps a list of nodes that are its neighbors in the ID space. These neighbors, together with short-cuts that link nodes with more distant nodes, are used to route lookups deterministically to their destination. The destination of a lookup is the node that is responsible for the requested content. Content or content descriptors are also hashed into the same ID space. A protocol specific rule assigns content IDs to nodes with certain IDs and the content or its description is stored at this responsible node.

Chord [1], e.g., uses a 1-dimensional ID space that is wrapped into a ring shape. Nodes are arranged with increasing IDs on the ring. A node is responsible for all content with IDs between its own ID and the ID of its predecessor on the ring. Each Chord node knows about its successor and predecessor on the ring and some short-cuts, so called fingers, to other nodes further away. Fingers are arranged in a way that the distance to the queried ID can at least be halved with every hop. A node's i^{th} finger is the first node succeeding the node ID plus 2^i:

$$i^{th}\ finger = successor(node\,ID + 2^i) \qquad (1)$$

Lookups are routed clockwise through the network using the finger entries. The lookup is always sent to the closest finger, which is still preceding the queried ID. If the query reaches the node directly preceding the ID on the Chord ring, this node forwards the request to its successor *s*. Node *s* is the direct successor of the ID and is therefore responsible for the queried content. Finalizing the lookup, *s* sends an answer with the queried ID back to the initiator of the lookup.

One of the most important tasks of structured P2P protocols is keeping up the overlay structure. This is even more important than providing an efficient search, as lookups can only be resolved if routing through the overlay is possible. If the overlay structure is corrupt in one part of the network, any route through that part of the topology will fail. A good P2P protocol is characterized by a reliable and efficient search. For structured networks this is only achievable in a highly stable topology. Stability comprises correctness of the neighbor entries as well as fast handling of topology changes, due to joining and leaving nodes. Especially in networks with high

Manuscript received June 30, 2005.

Gerald Kunzmann is working at the Institute of Communication Networks at the Munich University of Technology (TUM), 80333 Munich, Germany (phone: +498928923506; fax: +498928923523; e-mail: gerald.kunzmann@tum.de).

Andreas Binzenhöfer and Robert Henjes are working at the Department of Distributed Systems, Institute of Computer Science, University of Würzburg, 97074 Würzburg, Germany (e-mail: binzenhoefer@informatik.uni-wuerzburg.de, henjes@informatik.uni-wuerzburg.de).

churn rates, a fast, reliable and self-organizing stabilization algorithm is indispensable.

The basic Chord protocol has a simple stabilization mechanism. Each node *n* periodically sends a stabilization message to its successor *s*. Node *s* replies to this message by sending its predecessor entry *p* to node *n*. In the normal case, *p* should be equal to *n*. If a new node *j* joins between *n* and *s*, it contacts *s* and informs *s* that it is now participating in the network. Node *n* does not yet know about the existence of *j*. The next time *n* sends a stabilization message to *s*, *s* would return IP address and ID of its new predecessor *j*. Node *n* stores *j* as its new successor and, at the same time, informs *j* that it is its predecessor. This completes the stabilization algorithm, as the neighbor entries of all participating peers are pointing to the correct nodes again. A more detailed description of Chord can be found in [1].

If any node leaves the network, it just has to inform both its successor and predecessor, telling them their new predecessor and successor respectively, so that they can update their neighbor lists. If a node fails, due to e.g. a system crash or a discharged battery in a mobile device, it cannot inform its direct neighbors. So, the stabilization mechanism must handle failure events, too. We assume, that node *n*, lying between nodes *p* and *s*, fails. The next time node *p* runs stabilize, it receives no answer from node *n*. It may try sending stabilization messages several times, but after a certain timeout interval it must assume that node *n* is no longer participating in the network. The ring has broken apart, as node *n* knows about no other successor on the ring. This is why each node maintains a list of several successors. If the direct successor fails, it can drop that one from the list and try to connect to the next successor. Obviously, the list of successor entries also has to be updated regularly.

The costs of keeping up the overlay structure can be measured as the bandwidth required for stabilization messages, including join and leave messages. The next chapter describes how measuring the network stability can be accomplished.

Related work can be found in [4], where the correlation between network stability and required bandwidth for different structured P2P protocols is discussed. The authors analyze the effect of tuning different design parameters, but apply a constant churn rate, i.e. nodes crash and rejoin at exponentially distributed intervals with a mean of one hour. In this paper, we concentrate on the Chord protocol and exploit the influence of the same design parameters under a wide range of churn rates from mean session durations of two hours down to mean session durations of 10 minutes. Especially for high churn rates, i.e. short session durations, the basic Chord protocol is no longer able to establish a stable overlay topology. We present several modifications to Chord's stabilization protocol in order to make the resulting overlay structure more stable.

II. Analytical Approach

The efficiency of a stabilization algorithm can be evaluated by comparing all local neighbor lists with a global view of the network and counting the discrepancies. We distinguish between nodes that have wrong direct successors and nodes that have any wrong successor entries. Wrong direct successor and predecessor entries are worse than other entries in the neighbor list, as a node mainly communicates with its direct neighbors. All other neighbor entries are primarily for resilience purposes.

The costs of keeping up the overlay structure can be evaluated by measuring the used bandwidth required for signaling messages. Signaling traffic can be divided into three different cost types:

- Costs for keeping up the topology structure, including all messages that are sent while nodes join and leave the network, as well as all messages that are sent during stabilization.
- Costs for keeping the routing entries up to date.
- Costs for inserting, republishing and looking up content in the network (may be difficult to distinguish from the costs for keeping the routing entries up to date, as finger entries may be updated with information that is acquired during lookups, e.g. if recursive routing is applied).

In the context of this paper we only focus on costs required for keeping up the network structure. We analyze the efficiency of Chord's stabilization algorithm by measuring the number of nodes with erroneous neighbor lists and the resulting costs. We also vary different design parameters as well as the user behavior, i.e. their mean online durations.

Each simulation is run with a different set of parameters. To be able to evaluate the influence of each individual parameter apart from all other parameters, we do only change one parameter in each of our simulations. All other parameters are kept constant with these values:

- Number of Participants: 10.000
- Mean Online Duration: 60 Minutes
- Number of Successors: 5
- Stabilization Period: 30s

Our measurements were realized in an event-based simulator written in Ansi C. The simulations do assume a random, negative exponential distributed delay between two nodes in the overlay network. Packet loss is not considered.

The results acquired from the different simulation runs are discussed in the next chapter.

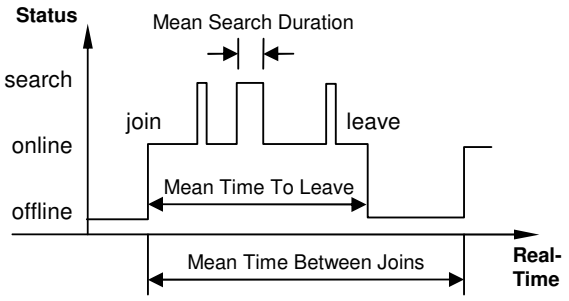

Fig. 1: A nodes lifetime consists of one or more sessions.

A node's lifetime can be described by its Mean Time Between Joins (MTBJ), its Mean Time To Leave (MTTL) and its Mean Search Duration (MSD) (see Fig. 1). The whole lifetime consists of one or more sessions, in which the node is participating in the network. Each session has an average length of MTTL and can be divided into active parts, where searches are performed, and passive parts. On average, nodes perform Mean Number of Searches with a Mean Search Duration (MSD) each in their online phases. After leaving the network or failing, a node is offline for a certain period (MTBJ – MTTL), and it rejoins MTBJ after its last join event. The average ring size, i.e. the average number of nodes being online at the same time, is:

$$ring\ size = number\ of\ users \cdot \frac{MTTL}{MTBJ} \qquad (2)$$

III. Results

A. Number of Participants

Structured P2P protocols have been developed to scale better than earlier protocols. Using a proactive routing scheme, it is possible to route lookups on one short and determined path to the node that is responsible for the requested content. Infrequent items can be resolved as good as popular content. Structured P2P approaches differ, amongst other things, in the number of hops required to resolve queries: from *O(logN)* [1,5,6] to *O(sqrt(N))* [7] to *O(1)* [8]. In this regard, Chord scales with *O(logN)*. As described in Chapter 1, lookups for content IDs are required in all processes, such as joining the network, finding new finger entries or inserting and searching content.

Structured P2P protocols also scale well with an increasing number of participants in terms of signaling traffic. Most algorithms require a constant bandwidth independent of the network size. Stabilization messages, e.g., are always exchanged between a node and its neighbor(s). Finger entries can be probed by a simple ping message. As the number of fingers and neighbors is independent of the number of peers participating in the network, the overhead per peer is constant.

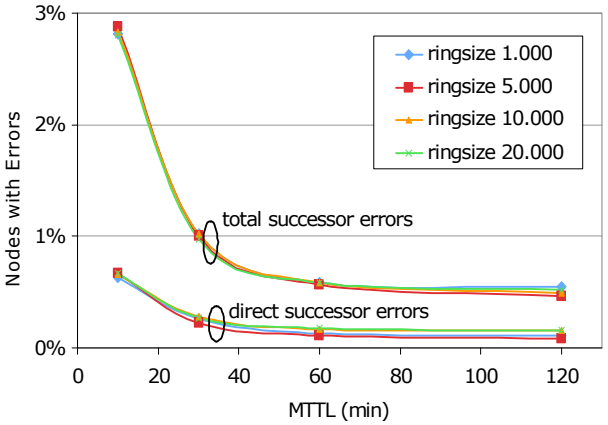

Fig. 2: The shorter the sessions, the more instable the overlay structure.

If hash functions with good stochastic properties are applied, a huge number of nodes actually smoothes the distribution of content over all nodes. Therefore, the probability that a node has to store a noticeable larger percentage of content or content descriptions is reduced. This is important as a node, responsible for more content, would otherwise have to respond to more lookups, keep more profiles up to date and so on.

Summarizing the above, structured P2P protocols do scale well with the number of participants in the network. Fig. 2 shows the results of our simulations. It proves that the number of peers with errors in their list of successors (upper 4 curves) or with wrong direct successors (lower 4 curves) is independent of the size of the overlay ring. The different curves for the different ring sizes almost coincide. Also, the required signaling overhead is nearly identical for different ring sizes, but increases with shorter MTTL values (see Fig. 3).

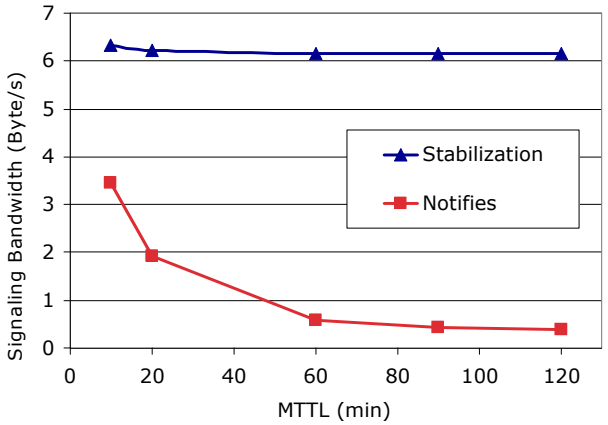

Fig. 3: The shorter the sessions, the more notifies are sent.

B. Mean online times (Churn Rate)

To be able to provide an efficient proactive routing, all routing tables have to be updated regularly. Each change in the overlay topology leads to erroneous entries in fixed routing tables. A node that joins the network, for example, has to be announced to all of its new neighbors in the Chord ring. Additionally, finger tables may provide a more efficient routing if the new node is inserted into them.

Nodes leaving the network have to send notification messages to all of their neighbors, whereas failed peers have to be detected by their former neighbors, which in turn have to make sure, that all outdated references are removed.

Each node causes its new neighbors on the ring to update their successor and predecessor lists when it joins the network, as well when it leaves or fails a certain time later. Thus, each node sets off two events, which change the overlay topology, per MTBJ.

$$Churn\ Rate = \frac{number\ of\ events}{time \cdot number\ of\ users} = \frac{2}{MTBJ} \tag{3}$$

$$Join\ Rate = \frac{number\ of\ joins}{time} \tag{4}$$

We define the churn rate as the average number of join and leave events per time interval per node (3). Note that the churn rate does only depend on the behavior of the end user and not on the size of the overlay. If, in a sample network, nodes are participating on average once a day, i.e. an MTBJ of one day, the churn rate per node would be 2/24 h^{-1} (3). If there are one million different users, the join rate would be 10^6 d^{-1}, i.e. about 41667 nodes join events per hour. If every node stays online for an average of two hours, i.e. an MTTL of 2 h, the average ring size would be about 83333 nodes (2).

The basic Chord protocol uses a periodic stabilization algorithm, i.e. each node checks periodically if it is still its successor's direct predecessor. If a node n has joined between a node p and its successor s, s returns the ID and IP address of its new predecessor n. However, if two or more nodes (n_1, n_2, … n_n) have joined between p and s, s would return node n_n instead of node p's new successor n_1. So, more than one stabilization period is necessary to repair all neighbor entries.

We can calculate the probability that more than one node joins between two neighbored nodes within one stabilization period by using series expansion.

$$P = 1 - \left(\frac{N!}{N^J \cdot (N-J)!} \right) \tag{5}$$

with $J = \frac{number\ of\ users}{MTBJ} \cdot stabilization\ period$

and $N = ring\ size$

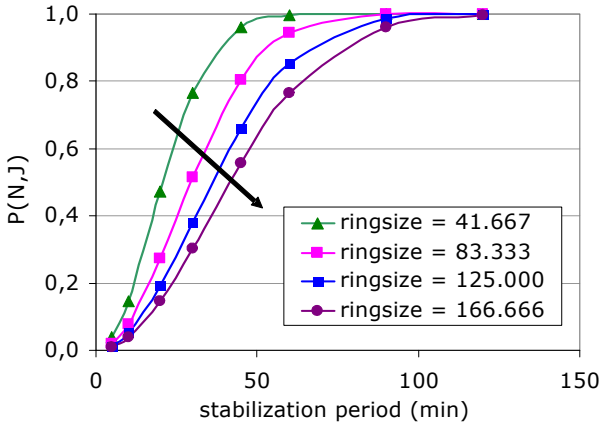

Fig. 4: The probability of two or more nodes joining between two neighbored nodes within one stabilization period increases with larger rings and longer stabilization periods.

If, e.g., in a network with 10 evenly distributed participants 2 nodes join, the ID of the second joining node can lie in 9 ID ranges where no other node has joined, or in the same ID range as the first joining node. Therefore, the probability of both nodes joining within the same ID range is 1/10. In our sample network with a total number of one million users, an MTBJ of one day and an MTTL of two hours we can calculate an average ring size of 83333 nodes using (2). If we assume a stabilization period of one minute, J is about 694 and P is about 0.9446 (4). Fig. 4 shows the correlation between P and the values ring size and stabilization period in the sample network with one million users and an MTBJ of one day.

We implemented a Chord variant, where each node not only stores several successors, but also several predecessor entries. Node s then knows about more predecessors and can return p's real successor (or at least a definitely closer node). Therefore, our stabilization algorithm can repair neighbor entries noticeably faster than the basic Chord algorithm. Each stabilize message contains a node's complete list of neighbors, so the receiving node can derive all necessary information from it. That is why it is not so disadvantageous if several nodes join between two neighbored nodes within one stabilization period.

Additionally, it is possible to send notification messages if a node observes a topology change in its neighborhood. If, for example, a new node joins the network and contacts its successor s for the first time, s could notify all of its neighbors about the new node. Then, almost all neighbor lists could be updated at once, without the need to wait for the next stabilization period. On the downside, each notification message increases the signaling bandwidth, but the stabilization period could be stretched in exchange, as the notification messages already update almost all neighbor lists. In scenarios with low churn rates, sending notification messages, together with a large stabilization period, would increase the topology's stability and reduce the required signaling bandwidth.

One drawback of notification messages is that they are sent at irregular points in time, which could lead to traffic peaks, whereas Chord's stabilization algorithm produces a constant bandwidth. If different notifies are sent within a short time and in a small part of the ring topology, e.g. if two or more nodes observe the same topology change at nearly the same time, the available bandwidth could be insufficient and packets would be lost. We approach this problem with the following rule: if a node observes that a new predecessor has joined the network or its old predecessor has failed, it sends notification messages to all nodes in its list of neighbors. If any other node observes a join/fail, it informs the successor of the newly joined/failed on his behalf. If several nodes observe the same join/fail event, only one set of notification messages will be sent.

As we can see in Fig. 2, the Chord protocol scales well in scenarios with moderate churn rates (more than 60 minutes mean session duration), but fails to scale in environments with high churn rates.

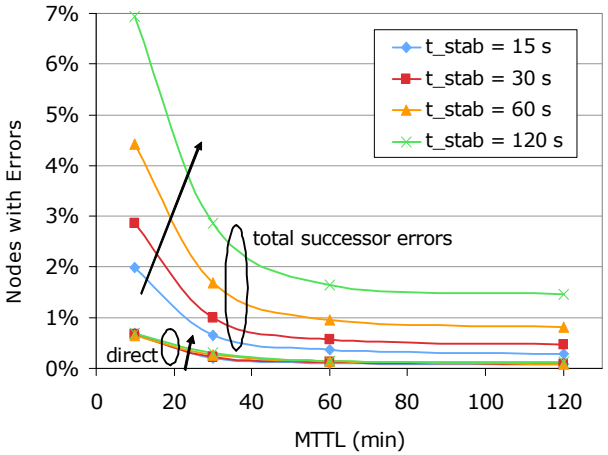

Fig. 5: Chord's stabilization period has a significant influence on the correctness of neighbor entries.

C. *Neighbor Stabilization Period*

As already mentioned in the previous subsection, the influence of Chords stabilization algorithm, especially the frequency of the stabilization calls, has a significant impact on the stability of Chords ring structure. Fig. 4 shows that the length of the stabilization period (t_{stab}) increases the probability that more than one join occurs within one stabilization period between two adjacent nodes. If node failures are also taken into account, the probability that two or more topology changes between closely neighbored nodes happen in a short period of time, is even higher.

Using a list of several predecessor and successor entries and sending notification messages can reduce the time that is necessary to repair all neighbor list entries. Still, a longer stabilization period leads to more neighbor list errors (see Fig. 5). The uppermost curve belongs to a stabilization period of 120 seconds. The upper 4 curves show the percentage of nodes with any error in their successor entries, whereas the lower 4 curves, which all coincide, show nodes with a direct successor error.

Sending complete neighbor lists can repair direct neighbor errors even for long t_{stab} values almost as good as for short values. In contrast, non-direct neighbor errors increase with longer stabilization periods. Reducing the stabilization period from 120s to 60s, for example, decreases the percentage of nodes with erroneous neighbor lists from 2.9% to 1.7% in a network with a mean session duration of 30 minutes. This dependency results from the fact that a node, that sends notification messages about a new/failed node *n*, does not know all other nodes that have *n* in their list of neighbors. As mainly more distant nodes are not informed about the change, it takes a while until the stabilization algorithm is able to correct all entries.

Again, bandwidth limitations prevent stabilize to be called with high frequencies. If t_{stab} is halved, stabilization requires double bandwidth and vice versa (not regarding notification messages). So, an optimal value for t_{stab} depends on the requirements of the system, the available resources and the user behavior (mainly MTTL).

D. *Number of Neighbors*

The main reason, on the one hand, to store more than one successor is that Chord's ring structure is lost as soon as one node looses all its successors. The probability of such a ring break is approximated in [9]. This probability gets smaller, the more neighbors a peer stores.

The main reason, on the other hand, to avoid a large set of neighbors is that the packet size of the stabilization messages grows with the number of neighbors (all neighbor entries are included in the stabilization packets). Furthermore, if notification messages are used, more notification packets have to be sent, as more neighbors must be informed about the change in the overlay topology.

If links fail with a high, but realistic, failure probability of pfail = 0.01, less than $\lceil \log_2(n) \rceil$ successors are sufficient to prevent a ring break with high probability [9]. In our simulations, five successor and five predecessor entries proved to be sufficient under realistic circumstances (see Fig. 6). Again, the upper 4 curves represent the percentage of nodes with any errors in their successor lists, and the lower 4 curves represent nodes with direct successor errors. We still recommend using a slightly larger set of neighbors in final implementations to prevent network break downs by all means.

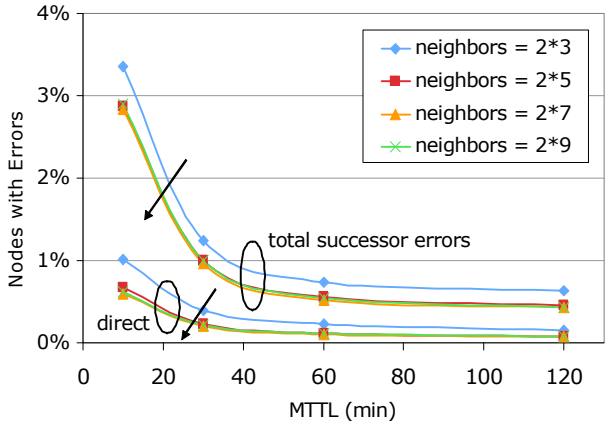

Fig. 6: Using more than 2*3 neighbors does not significantly improve the stability of the overlay topology.

E. *Finger Update Period*

We define the finger update period as the time wherein all finger entries are update once. Therefore, the more fingers each node stores, the more finger updates are done in every finger update period. Updating fingers more frequently requires more bandwidth, but may be reasonable in scenarios with high churn rates as out-dated routing entries result in timeouts and therefore in increased search durations. So updating fingers more often would increase the efficiency of the system.

However, the finger update period does hardly affect the stability of Chords ring structure as stabilization entries (neighbors) are kept completely separate from routing entries (fingers) in our current implementation. Yet, we are working on combining finger and neighbor entries. As [9] proves, in a network with an identifier space of m bits, each node maintains only $j = O(log_2(N))$ actually different finger entries. All other $m - j$ finger entries coincide with the nodes direct successor, and even more fingers lie within the range of the node's successor list.

If the routing algorithm would consider the successor entries for routing lookups through the network, it could keep a clearly shorter list of fingers, dropping entries that are already stored as one of the node's successors. This means that only $O(log2(N))$ fingers have to be kept up-to-date instead of m. That leads to a reduced bandwidth as fewer fingers are updated within one finger update period, or to more up to date routing entries, if the finger update period is reduced and the bandwidth is kept constant.

IV. Conclusion

In this paper we showed that an application based on Chord is feasible in huge networks. Networks with high churn rates, on the other hand, may be an obstacle for structured P2P protocols, as the signaling overhead grows with shorter session durations. At some point, flooding the network, without building up a deterministic network structure, may be more efficient in terms of costs. We are going to evaluate this aspect in some future work. Still, structured P2P protocols yield the advantage of being able to always find content, even if it is rare or unique.

We have analyzed the influence of different design parameters on the stability of the Chord structure. The number of neighbors can be kept small without making the topology noticeably less stable. Tuning the stabilization period to small values is, in contrast, necessary for networks with high churn rates. This knowledge helps application developers to set their parameters to optimal values for their application environment. We have also mentioned some changes to Chords stabilization algorithm, so that it consumes less signaling overhead and, therefore, can handle higher churn rates. At the moment, we are working on a token based stabilization mechanism that makes Chord even more stable without increasing the necessary signaling bandwidth [10].

References

[1] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," presented at ACM SIG-COMM Conference, 2001.

[2] RFC-Gnutella 0.6, http://rfc-gnutella.sourceforge.net/developer/testing/, 2002

[3] C. Bettstetter et al. "Self-Organization in Communication Networks: Overview and State-of-the-Art," Wireless World Research Forum (WWRF) Special Interest Group (SIG) 3, White Paper 2005.

[4] J. Li, J. Stribling, T. M. Gil, R. Morris, and M. F. Kaashoek, "Comparing the Performance of Distributed Hash Tables under Churn," presented at 3rd International Workshop on Peer-to-Peer Systems (IPTPS), San Diego, CA, 2004.

[5] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Ob-ject Location and Routing for Large-Scale Peer-to-Peer Sys-tems," presented at IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001.

[6] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System based on the XOR Metric," presented at International Workshop on Peer-to-Peer Systems (IPTPS), 2002.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," presented at ACM SIGCOMM Conference, 2001.

[8] I. Gupta, K. Birman, P. Linga, A. Demers, and R. v. Renesse, "Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead," presented at Second International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA, USA, 2003.

[9] A. Binzenhöfer, D. Staehle, and R. Henjes, "On the Stability of Chord-based P2P Systems," University of Würzburg 347, 2004.

[10] G. Kunzmann, R. Nagel, J. Eberspächer, "Increasing the reliability of structured P2P networks," presented at 5th International Workshop on Design of Reliable Communication Networks (DRCN), 2005