SICS Technical Report
T2008:03

ISRN : SICS-T--2008/03-SE
ISSN : 1100-3154

An Analytical Framework for the Performance Evaluation of Proximity-aware Structured Overlays

by

Supriya Krishnamurthy and John Ardelius

Swedish Institute of Computer Science
Box 1263, SE-164 29 Kista, SWEDEN

An Analytical Framework for the Performance Evaluation of Proximity-aware Structured Overlays *

Supriya Krishnamurthy[1,2] and John Ardelius[1]
[1] Swedish Institute of Computer Science (SICS), Sweden
[2] IMIT, KTH-Royal Institute of Technology, Sweden
{supriya,john}@sics.se

February 28, 2008

Abstract

In this paper, we present an analytical study of proximity-aware structured peer-to-peer networks under churn. We use a master-equation-based approach, which is used traditionally in non-equilibrium statistical mechanics to describe steady-state or transient phenomena. In earlier work we have demonstrated that this methodology is in fact also well suited to describing structured overlay networks under churn, by showing how we can accurately predict the average number of hops taken by a lookup, for any value of churn, for the Chord system. In this paper, we extend the analysis so as to also be able to predict lookup latency, given an average latency for the links in the network. Our results show that there exists a region in the parameter space of the model, depending on churn, the number of nodes, the maintenance rates and the delays in the network, when the network cannot function as a small world graph anymore, due to the farthest connections of a node always being wrong or dead. We also demonstrate how it is possible to analyse proximity neighbour selection or proximity route selection within this formalism.

1 Introduction

Structured peer-to-peer networks have in recent years emerged as a promising infrastructure in large scale distributed applications. The concept of having a distributed hash table, DHT, among participating nodes [8] together with a structured lookup protocol forms the basis of many *overlay networks* [18, 19, 22, 23, 25]. Structured overlays have turned out to exhibit interesting properties in that many implementations allow a node to find a specific key in a number of network hops logarithmic in system size (keys or

*This work is funded by the 6th FP EVERGROW project.

nodes), resulting in very good scalability. The number of network hops is however not the only quantity of interest. Design of structured overlay networks has to consider tradeoffs between various performance meassures such as the number of connections kept between the participating nodes (state) vs. expected number of hops to find a specific target (hop count), how often a node needs to repair broken links (maintanence) vs. the number of nodes joining and leaving the system (churn) etc.

Since peer-to-peer systems lack a central management point, global system properties can only be affected through change in local policies and actions. How to predict the impact of local policies on the overall network performance is an important but complex issue.

A few attempts have been made to theoretically analyze overlay networks under realistic dynamic conditions [4, 6, 11] but in order to fully understand dynamic peformance aspects more elabortate models are needed.

In this work we analytically study the impact of link delays and proximity selection in the Chord [23] system under churn. The results show that there is a critical point in the parameter space at which the system with high probability breaks down, in that it can no longer efficiently route searches. With high enough churn rate and large link delays the nodes are not able to maintain consistent state, a finite fraction of the connections are always wrong or dead. The results are not unique for Chord but are generalizable to any structured overlay. These kind of feed back based phase transitions are common in many kinds of complex systems and our results are only begining to explore their role in distributed computer systems.

The paper is organized as follows. We describe related work in Section 2, and our analytical framework in Section 3. Section 4 briefly introduces all the parameters in the model. Section 5,6, 7 and 8 describe our results followed by a summary in section 9.

2 Related work

Theoretical models of overlay networks for proving bounds on the number of expected network hops often do not consider practical problems such as network delays, massive node failures and link congestion. Surveys of the impact of link delays in real large scale overlay networks do exist [5] and numerical experiments also show that there is a tradeoff between cost and performance in overlay networks [14]. Different proximity-aware lookup policies have been suggested to increase performance and some are numerically evaluated in [3, 20]. Analytic work on proximity-aware lookup policies is to the best of our knowledge still lacking but related work exists. In [9] a model of congestion-aware routing is evaluated and analytical models for network resilience to failures are studied in [4, 24].

3 Analytical Framework of the Master-Equation Approach

In a typical P2P network, there are many interleaved processes happening in time. Nodes join or fail. Maintenance messages are scheduled, requests for lookups are sent out to different nodes, and queries are answered. Apart from these processes, there are also several other factors that affect the performance of the network. These include the number of nodes currently in the network, the specific lookup or maintenance algorithm used, the delays in receiving replies to queries, the number of contacts per node and how these contacts are chosen in the first place. There are clearly many parameters that can be varied and the performance of the network, usually measured in terms of lookup latency, depends on each and every one of them. The question then arises, of whether there is a systematic and modular way to quantify the effect of different design choices and/or different parameters. Currently extensive simulation is the prevalent tool for gaining such knowledge. Examples include the work of Li *et al.* [15], Rhea *et al.* [21], and Rowstron *et al.* [2].

There has also been a lot of theoretical analysis done on DHT's. For instance, Liben-Nowell *et al.* [16] prove a lower bound on the maintenance rate required for a network to remain connected in the face of a given churn rate. Aspnes *et al.* [1] give upper and lower bounds on the number of messages needed to locate a node/data item in a DHT in the presence of node or link failures. The value of theoretical studies of this nature is that they provide insights neutral to the details of any particular DHT. However performance can vary substantially within these bounds. In addition, it is usually not clear how to extend these results if a design choice or a parameter is varied.

We thus adopt a different approach to the theoretical analysis of DHT's. We concentrate not on establishing bounds, but rather on a more precise prediction of the relevant quantities in such dynamically evolving systems. Our analysis is based mainly on the Master-Equation approach used in the analysis of physical systems. We have previously introduced our approach in [11–13] where we have presented a detailed analysis of the Chord system [23]. More specifically, we have shown how we can quantify the number of hops taken by lookups in Chord, for any value of churn (we specify how exactly we quantify churn a little later in this section), for any value of the number of nodes in the network, and for any number of connections per node. As a bonus, we see that the analysis has a modular form in the sense that if a design choice (such as the specific maintenance strategy that a node uses to keep its connections from getting outdated) is varied, we do not need to rethink the whole analysis again. It is enough to redo the relevant part which is affected by the design choice. We find that a very important quantity which is affected substantially by the design choice is the fraction of dead (or corrupted) network links. This depends not only on the specific maintenance strategy used [13], but also on the manner these connections are constructed. In addition it also depends on churn (clearly) and even on which specific connection we are talking about (to a neighbor, or to a node far away and if the latter, then exactly how far away). This quantity goes as an input into the Lookup equation, solving which enables us to predict the average number of hops for any system parameter. The structure of the Lookup Equation itself, depends only on the routing algorithm used as well as the specific overlay graph, but it requires as an input both the fraction of dead fingers as well as the distribution of node id's in the key space.

In this paper, we demonstrate for the first time to our knowledge, how to generalise the master equation approach to take *proximity* into account. We first show how we can take delays into account, by evaluating the lookup latency for a given average delay in the network. An interesting feature that comes out of this analysis is that there is a *phase transition* in the space of the parameters. In one region of the parameter space, the system is viable, and a given node has, with some probability, all its connections functioning at least part of the time. In a different region of the parameter space however, the furthest connections that a node has (its largest fingers in Chord) are always dead, and even if the ring is not disconnected, the system no longer functions like a 'small-world'. We then use this analysis to predict the performance of the network for the two different choices of Proximity Neighbor Selection (PNS) and Proximity Route Selection (PRS). The utility of our approach is thus, to develop an analytical framework for thinking about proximity-aware networks. While the numbers we get from our analysis will certainly depend on the slightest details of the implementation of Chord, the qualitative picture should certainly hold. In addition, our approach is both detailed (in taking the *minutiae* of system protocols into account) and broad. A similar approach should work for almost any DHT on the market.

4 The Churn Model and Parameters defining the system

Due to space limitations, we provide only a very brief description of Chord below, with a view to explaining the parameters involved.

The Chord Ring is a circular key space of size $\mathcal{K}$ populated by N nodes.

Fingers. To reduce the average lookup path length, nodes keep $\mathcal{M} = \log_2 \mathcal{K}$ pointers known as the "fingers". Using these fingers, a node can retrieve any key in $O(\log N)$ hops. The fingers of a node n (where $n \in 0 \cdots \mathcal{K}-1$) point to exponentially increasing distances of keys away from n. That is, $\forall i \in 1..\mathcal{M}$, n points to a node whose key is equal to $n + 2^{i-1}$. The corresponding entry in the finger table is the successor of this key.

Stabilization, Delays, Churn & Steady State. To keep the pointers up-to-date in the presence of churn, we consider a periodic stabilization strategy (We have also analysed a reactive strategy in [13]). We define λ_j as the rate of joins per node, λ_f the rate of failures per node and λ_s as the rate at which stabilizations are scheduled, per node. With probabilty α, the node chooses to stabilise a successor and with probability $1-\alpha$, it sends out a lookup message for any one of its finger. One can think of these rates as the inverse of the average time-interval between a join, fail or stabilisation event happening for any given node on the ring. In our analysis, we do assume that these rates are independent of time, so that we can impose the steady-state condition $\lambda_j = \lambda_f$ at all times unless otherwise stated. Further it is useful to define $r \equiv \frac{\lambda_s}{\lambda_f}$ which is a relevant measure of churn, e.g, $r = 50$ means that a join/fail event takes place every half an hour for a stabilization which takes place once every 36 seconds. To take delays into account, we have another rate λ_t which we take to be the rate of message transmission, or the inverse of the average time taken for a message to reach the recipient, over one overlay link. In our analysis in Section 5, λ_t can be thought of as an average rate of transmission or the inverse of the average delay of a link in the network. In our section on PNS and PRS, we also need a distribution of delays in the network. For ease of analysis, we use a simple model where there are only two kinds of links, good or bad. The bad links have transmission rate $x\lambda_t$, while the good ones transmit at simply λ_t. Throughout the paper we will use the terms $\lambda_j N\Delta t$, $\lambda_f N\Delta t$, $\alpha\lambda_s N\Delta t$, $(1-\alpha)\lambda_s N\Delta t$ and $\lambda_t N\Delta t$ to denote the respective probabilities that a join, failure, initiation of a successor stabilization, the initiation of a finger lookup or a message transmission across a link takes place anywhere on the ring during a micro period of time of length Δt.

5 Results

We are interested in analysing a model where messages sent across overlay links are not necessarily instantaneous. The delays could be caused by several factors. For one, a node may take some time after the receipt of a message, before sending it, if there is a long queue.For another, the underlying network may impose delays.

These are typically in the range of 100 ms to 3 seconds.

The delay gives us a typical time between the message being sent by the sender and the message being received by the recipient. Our model for the delay in a given link is the following:

If the delay is 100 ms we assume $\lambda_t = 1/100$, which means that in a time $\Delta t = 100ms$, the probablilty that a message gets relayed is 1 but there is some small probability that a message gets transmitted earlier. This idea of

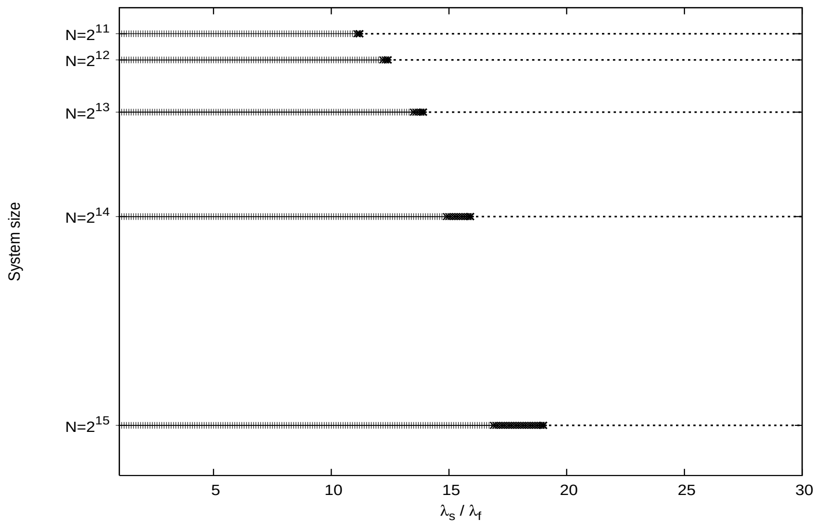

Figure 1: Different regions of stability as a funtion of λ_s/λ_f, which is the ratio of the rate at which stabilisation actions are scheduled (though not necessarily executed, because of delays) to the rate of failures of nodes. For low rates the ring breaks up. For slightly higher value the ring is connected but the delays cause the largest fingers of any node, to be dead all the time. The existence of this region is one of the principle predictions of our formalism. For higher rates of stabilisation, the system functions properly. As can be seen the region of instability increases with the number of nodes in the network. For this plot it is assumed that the time taken for the messages to be delivered across one overlay link is 100 times faster than the time interval between failures.

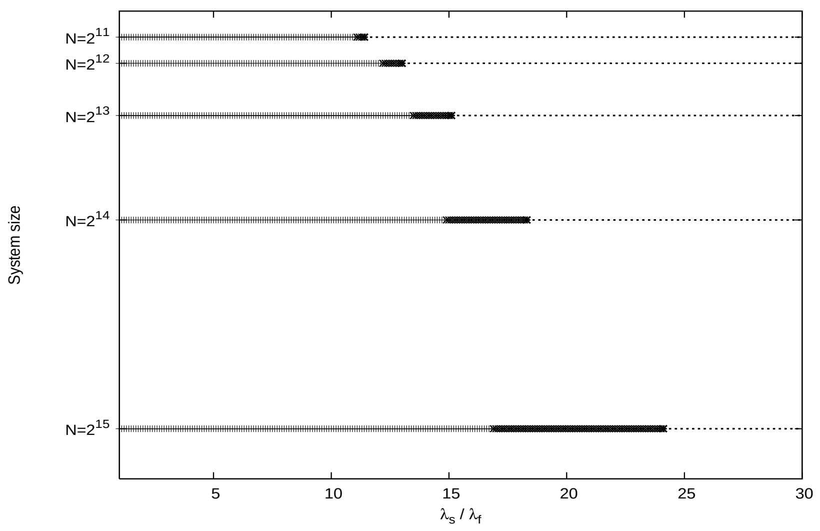

Figure 2: The region in which the system does not function also increases very rapidly with an increase in the average delay in the network. For this plot, the time taken for the messages to be delivered across one overlay link is only 19 times faster than the time interval between failures. The unstable region for the largest system size is almost 4 times as much as in Fig. 1.

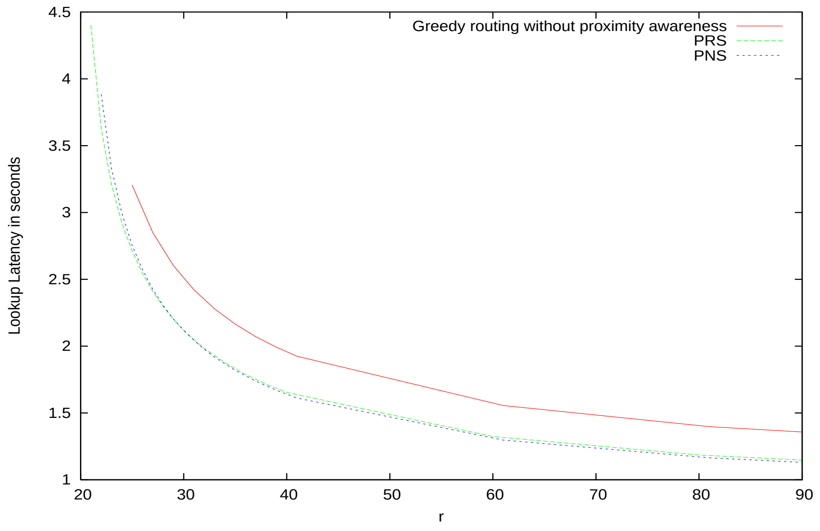

Figure 3: The average lookup latency in seconds, versus $r = \lambda_s/\lambda_f$, for the three schemes studied in this paper for $N = 23000$. This is in the region of parameter space when the system is functioning normally.

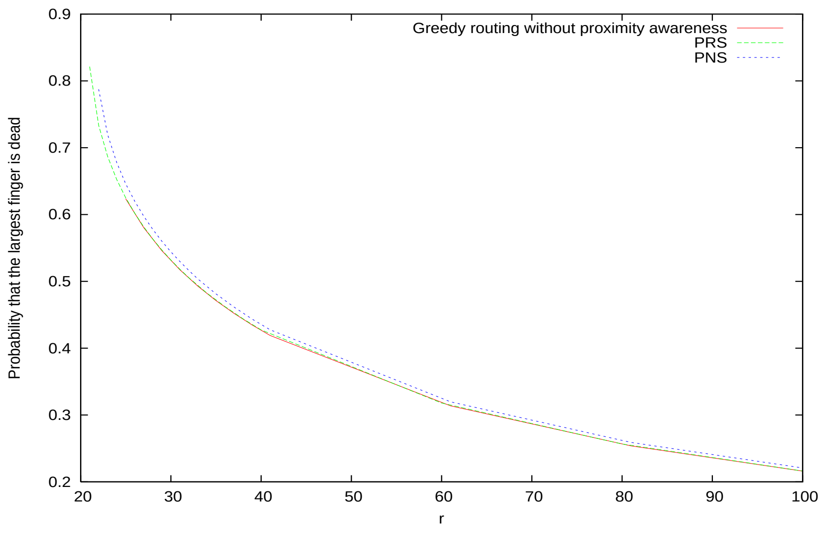

Figure 4: The probability of the largest finger being dead versus $r = \lambda_s/\lambda_f$, for the three schemes studied in this paper (same N value as in Fig 3.), in the region of parameter space when the system is functioning normally.

converting time into rates is the same as explained in section 4.

In general, different network connections have different delays and there will be a distribution of delays.

For the purposes of the first model that we study in this paper, we only need the *average* delay per link, λ_t.

When we consider PRS and PNS in the following sections, the distribution of delays will also be important. In this case, for ease of analysis, we will consider the following simple distribution:

A fraction $(1-p)$ of the links are "good " links with a delay of 100 ms. A fraction p of the links are "bad" with a delay which can range from 300 ms to 3 s.

Let us now consider how we go about analysing such a model.

- Given a ring of $\mathcal{K}$ keys and N nodes (on average), where nodes can join and leave independently, the probability that two adjacent nodes are a distance x apart on the ring is simply $P(x) = \rho^{x-1}(1-\rho)$ where $\rho = \frac{\mathcal{K}-N}{\mathcal{K}}$. Using this distribution, it is easy to estimate the probability that there is definitely at least one node in an interval of length x. This is: $a(x) \equiv 1 - \rho^x$. The probability that the *first* node encountered from any key is at a distance i from that key is then $b_i \equiv \rho^i(1-\rho)$. Hence the conditional probability that the first node from a given key is at a distance i *given* that there is at least one node in the interval is $bc(i,x) \equiv b(i)/a(x)$.

- The probability that the first successor is wrong (which means lookups are inconsistent) follows along the lines of our earlier calculations [11–13].

- We now come to the probabilities of fingers being dead. Delays play an important role in the correction of dead fingers. This is because, in the Chord protocol, new information about fingers is received through lookups. And how long lookups take, as well as the probability of lookups being executed sucessfully depends on the delays in the network, as we will see below.

The fraction of failed fingers, in its turn, affects the number of hops that lookups take, since failed fingers cause timeouts and hence increase the lookup length.

So in contrast to the situation without delays (where lookups were considered to take place many orders of magnitude faster than the time-scale of node joins, failures and stabilisation and hence did not play any role in the fraction of dead fingers), here we have a system with feedback. Delays affect the probabilites that lookups are succesful and hence the probability that dead fingers are successfully corrected. Dead fingers increase the number of hops that lookups take. Hence for high churn, we could have a situation where lookups take longer, fingers are corrected slower, which causes lookups to take yet longer. Depending on the value of churn, we find that there is a region of parameter space in which the system is just not viable. That is for some values of joins and failures, N and average delay time, the probabilities that larger fingers are dead goes to 1 and hence the system does not function efficiently as a DHT anymore (Fig. 1 and 2).

In the following paragraphs, we detail the dependance of the probabilites of dead fingers on the lookup length and vice versa.

For the moment, we only consider fingers being *dead* not wrong. Unlike members of the successor list, *alive* fingers even if outdated (which in Chord, means that instead of pointing to the first successor of an interval, sometimes the finger maybe pointing to the second or third node. This happens only rarely though, if $\lambda_s >> \lambda_j$) always bring a query closer to the destination. Hence they do not affect the consistency of a lookup or even substantially the lookup length.

We make an optimisation in the correction of the earlier fingers. On average $ln_2(K) - ln_2(N)$ of them are basically just the first successor. So we assume that these fingers are corrected every time the first successor is corrected and hence have the same probability of being dead.

For the remaining $ln_2(N)$ fingers, we assume that when the node decides to schedule a finger correction, it sends a lookup for any one of these $ln_2(N)$ fingers randomly.

Fingers of nodes far apart are independant of each other. Fingers of adjacent nodes can be correlated and we take this into account.

Let $f_k(r, \alpha)$ denote the fraction of nodes whose k^{th} finger points to a failed node and $F_k(r, \alpha)$ denote the respective number. For notational simplicity, we write these as simply F_k and f_k. We can predict this function for any k by estimating the gain and loss terms for this quantity, caused by a join, failure or stabilization event, and keeping only the most relevant

Table 1: The relevant gain and loss terms for F_k, the number of nodes whose k^{th} fingers are pointing to a failed node for $k > 1$.

$F_k(t+\Delta t)$	Probability of Occurence
$= F_k(t)+1$	$c_{3.1} = (\lambda_j N \Delta t)\sum_{i=1}^{k} p_{join}(i,k) f_i$
$= F_k(t)-1$	$c_{3.2} = (1-\alpha)\frac{1}{ln_2(N)} f_k(\lambda_s N\Delta t) a^{c_k} \frac{\lambda_t}{\lambda_t+\lambda_s c_k}$
$= F_k(t)+1$	$c_{3.3} = (1-f_k)^2[1-p_1(k)](\lambda_f N \Delta t)$
$= F_k(t)+2$	$c_{3.4} = (1-f_k)^2(p_1(k)-p_2(k))(\lambda_f N\Delta t)$
$= F_k(t)+3$	$c_{3.5} = (1-f_k)^2(p_2(k)-p_3(k))(\lambda_f N\Delta t)$
$= F_k(t)$	$1-(c_{3.1}+c_{3.2}+c_{3.3}+c_{3.4}+c_{3.5})$

terms. These are listed in Table 1.

The calculations of the probability of the k^{th} finger being dead follows along the lines of our earlier work [11–13], with a few essential differences in the term $c_{3,2}$. To understand the probability that the k^{th} finger is corrected in time Δt, a node with a dead k^{th} pointer needs to have sent out a lookup (this happens with probability $(1-\alpha)\frac{1}{ln_2(N)} f_k(\lambda_s N\Delta t)$ and this lookup needs to be succesful. To estimate the latter probability, let n_k be the number of hops for one particular lookup message for the k^{th} finger with a given value of churn. We can think of this as a series of n_k messages $\rightarrow \{m^{n_k}, m^{n_{k-1}}...m^1\}$.

The exact manner in which the lookup approaches the target depends on the routing algorithm and is explained in the lookup equation.

To estimate the probability that a lookup is succesful however, we need to consider how a message succesfully makes a transition from m^{n_k} to $m^{n_{k-1}}$. Let us use M^{n_i} to denote the number of nodes which have to relay messages m^{n_i} forward. Then

$$M^{n_i} \rightarrow \begin{cases} M^{n_i}+1 & ;(\lambda_t\Delta t)M^{n_i+1} \\ M^{n_i}-1 & ;(\lambda_f\Delta t)M^{n_i} \\ M^{n_i}-1 & ;(\lambda_t\Delta t)M^{n_i} \end{cases} \tag{1}$$

The explanation of the terms is as follows. The number of nodes with message m^{n_i} increases by one, if a successful transmission takes place from a node with $m^{n_{i+1}}$ and decreases by one either if a node carrying the message fails or if it successfully transmits it.

In steady state:

$$M_{n_i} = M_{n_{i+1}}[\frac{\lambda_t}{\lambda_t+\lambda_f}] \tag{2}$$

hence,

$$M_1 = [\frac{\lambda_t}{\lambda_t+\lambda_f}]^{n_k} \tag{3}$$

We define

$$\frac{\lambda_t}{\lambda_t + \lambda_f} = a \tag{4}$$

which explains term $c_{3,2}$ partially. The important variable here is the ratio of the rate of delays to the rate of failures. To consider some numbers, if $\lambda_t \approx 1/100\text{ms}$ and $\lambda_f \approx 1/3\text{mins}$ then $a = \frac{1800}{1801} = 0.999$. If $a = 1$ we get the case without delays studied earlier [11–13]. If the delay is larger however (either due to queueing or congestion), a could be much less than one.

We study $0.9 < a < 1$ in our system. Since the ratio $\frac{\lambda_t}{\lambda_f}$ is the relevant quantity in our analysis, a 100ms delay with a node joining or failing every 3 minutes is equivalent to a 1s delay if a node joins or fails every 30 minutes. The "time" that lookups take will of course be larger in the latter case. But the *probability* that lookups are successful is not affected.

The last term in $c_{3,2}$ is the ratio $\frac{\lambda_t}{\lambda_t + \lambda_s n_k}$. This multiplied by λ_s is the *effective* rate at which correction messages ,which took n_k hops, are received. Since even a successful lookup takes time and queries are not answered instantaneously. c_k is the average number of hops that a lookup for the k^{th} finger takes.

The equation we get for the f_k's is hence

$$(1 - f_k)^2[1 + \tilde{p}_{rep}] + p_{join}(k)f = \frac{(1-\alpha)}{ln_2 N} f_k \lambda_s \sum a^{n_k} p(n_k) \frac{\lambda_t}{\lambda_t + n_k \lambda_s} \tag{5}$$

In the right hand side we have $a^{n_k} p(n_k) \frac{\lambda_t}{\lambda_t + n_k \lambda_s}$ summed over n_k, where $p(n_k)$ is the probability that a lookup for the k^{th} finger takes n_k steps. We approximate the term inside the sum by $a^{c_k} \frac{\lambda_t}{\lambda_t + c_k \lambda_s}$, where $c_k = \sum n_k p(n_k)$. This is justified for the parameter values of interest $[a \to 1]$ and if $p(n_k)$ is a distribution with a well defined average and small standard deviation (as seems to be the case [7, 17]).

We should mention that the master equation formalism can be used to estimate the error in this estimate and in fact also the distribution $p(n_k)$. However in this paper, we only use

$$c_k = \sum n_k p(n_k) \tag{6}$$

as a measure of the performance of the system.

The equation for f_k is hence:

$$(1 - f_k)^2[1 + \tilde{p}_{rep}] + p_{join}(k)f = \frac{(1-\alpha)}{ln_2 N} f_k \lambda_s a^{c_k} \frac{\lambda_t}{\lambda_t + \lambda_s c_k} \tag{7}$$

If $a = 1$, or λ_t is infinitely large, the above is a simple quadratic equation and we get back our earlier results. However if $a \neq 1$ we need the value of c_k.

As we will see below, c_k itself also depends on f_k. Hence the equation for c_k and f_k have to be solved simultaneously and consistently to obtain the values for both c_k and f_k.

6 Lookup equation

In this section, we demonstrate how the information about the failed fingers and successors can be used to predict the cost of stabilizations, lookups or in general the cost for reaching any key in the id space. By cost we mean the number of hops needed to reach the destination *including* the number of timeouts encountered en-route. For this analysis, we consider timeouts and hops to add equally to the cost, though we could as easily study this as an additional parameter as well.

Define $C_t(r, \alpha)$ (also denoted by C_t) to be the expected cost for a given node to reach some target key which is t keys away from it (which means reaching the first successor of this key). For example, C_1 would then be the cost of looking up the adjacent key (1 key away).

To find the expected cost for reaching a general distance t we need to closely follow the Chord protocol, which would lookup t by first finding the closest preceding finger. For the purposes of the analysis, we will find it easier to think in terms of the closest preceding *start*. Let us hence define ξ to be the *s*tart of the finger (say the k^{th}) that most closely precedes t. Hence $\xi = 2^{k-1} + n$ and $t = \xi + m$ *i.e.*, there are m keys between the sought target t and the start of the closest preceding finger. With that, we can write a recursion relation for $C_{\xi+m}$ as follows:

$$
\begin{aligned}
&C_{\xi+m} = C_\xi \left[1 - a(m)\right] \\
&+ (1 - f_k)a(m)\left[1 + \sum_{i=0}^{m-1} bc(i,m)C_{m-i}\right] \\
&+ f_k a(m)\Bigg[1 + \sum_{i=1}^{k-1} h_k(i) \\
&\sum_{l=0}^{\xi/2^i - 1} bc(l, \xi/2^i)(1 + (i-1) + C_{\xi_i - l + m}) + O(h_k(k))\Bigg]
\end{aligned} \tag{8}
$$

where $\xi_i \equiv \sum_{m=1,i} \xi/2^m$ and $h_k(i)$ is the probability that a node is forced to use its $k - i^{th}$ finger owing to the death of its k^{th} finger. The probabilities a, b, bc have already been introduced earliler and we define the probability $h_k(i)$ below.

The lookup equation though rather complicated at first sight merely accounts for all the possibilities that a Chord lookup will encounter, and deals with them exactly as the protocol dictates.

The first term accounts for the eventuality that there is no node intervening between ξ and $\xi + m$ (occurs with probability $1 - a(m)$). In this case, the cost of looking for $\xi + m$ is the same as the cost for looking for ξ.

The second term accounts for the situation when a node does intervene in between (with probability $a(m)$), and this node is alive (with probability $1 - f_k$). Then the query is passed on to this node (with 1 added to register the increase in the number of hops) and then the cost depends on the length of the distance between this node and t.

The third term accounts for the case when the intervening node is dead (with probability f_k). Then the cost increases by 1 (for a timeout) and the query needs to find an alternative lower finger that most closely precedes the target. Let the $k - i^{th}$ finger (for some i, $1 \leq i \leq k-1$) be such a finger. This happens with probability $h_k(i)$ *i.e.*, the probability that the lookup is passed back to the $k - i^{th}$ finger either because the intervening fingers are dead or share the same finger table entry as the k^{th} finger is denoted by $h_k(i)$. The start of the $k - i^{th}$ finger is at $\xi/2^i$ and the distance between $\xi/2^i$ and ξ is equal to $\sum_{m=1,i} \xi/2^m$ which we denote by ξ_i. Therefore, the distance from the *start* of the $k - i^{th}$ to the target is equal to $\xi_i + m$. However, note that $fin_{k-i}.node$ (which is the node which first precedes $fin_{k-i}.start$ the start of the $k - i^{th}$ region) could be l keys away (with probability $bc(l, \xi/2^i)$) from $fin_{k-i}.start$ (for some l, $0 \leq l < \xi/2^i$). Therefore, after making one hop to $fin_{k-i}.node$, the remaining distance to the target is $\xi_i + m - l$. The increase in cost for this operation is $1 + (i - 1)$; the 1 indicates the cost of taking up the query again by $fin_{k-i}.node$, and the $i - 1$ indicates the cost for trying and discarding each of the $i - 1$ intervening fingers. The probability $h_k(i)$ is easy to compute given the distribution of the nodes and the f_k's computed in the previous section.

$$\begin{aligned} h_k(i) =& a(\xi/2^i)(1 - f_{k-i}) \\ & \times \Pi_{s=1,i-1}(1 - a(\xi/2^s) + a(\xi/2^s) f_{k-s}), i < k \\ h_k(k) =& \Pi_{s=1,k-1}(1 - a(\xi/2^s) + a(\xi/2^s) f_{k-s}) \end{aligned} \tag{9}$$

In Eq. 9 we account for all the reasons that a node may have to use its $k - i^{th}$ finger instead of its k^{th} finger. This could happen because the intervening fingers were either dead or not distinct. The probabilities $h_k(i)$ satisfy the constraint $\sum_{i=1}^{k} h_k(i) = 1$ since clearly, either a node uses any one of its fingers or it doesn't. This latter probability is $h_k(k)$, that is the probability that a node cannot use any earlier entry in its finger table. In this case, n proceeds to its successor list. The query is now passed on to the first alive successor and the new cost is a function of the distance of this node from the target t. We indicate this case by the last term in Eq. 8 which is $O(h_k(k))$. This can again be computed from the inter-node distribution. However in practice, the probability for this is extremely small except for targets very close to n. Hence this does not significantly affect the value of general lookups and we ignore it in our analysis.

The cost for general lookups is hence

$$L(r, \alpha) = \frac{\Sigma_{i=1}^{\mathcal{K}-1} C_i(r, \alpha)}{\mathcal{K}}$$

We solve the lookup equation numerically and search for a value of f_k which satisfies the f_k equation. We need to do this for all k (for the different fingers).

Results are shown in Figures $1 - 4$.

Figures 1 and 2 show the region in parameter space when the phase transition takes place and the system is no longer viable (the thick line). Below this point , there is no solution for the coupled f_k-c_k equations. $f_k \rightarrow 1$ and the network can no longer rely on having fingers in every sector. The difference in the two figures is the average delay in the network. In the first case, the time taken to send a message across a link is taken to be 100 times faster than the average session time of nodes. In the second case, it is 19 times faster. As can be seen, the system is very sensitive to such an increase.

Figures 3 and 4, show the average lookup length and f_k (the probability that the largest finger is dead), respectively for a proximity unaware system with delays using plain greedy routing, and a system with delays using PNS or PRS (the specific algorithms we use for PNS and PRS are elaborated on in the next two sections). As can be seen in Fig. 3, for a system with delays, using PNS is very slightly better than PRS (as discussed in [5]) when r is large (which is equivalent to low churn), but the situation reverses when churn is high. Using either of them is certainly better than using simply the greedy lookup strategy without any proximity awareness.

Interestingly from Figure 4, we see that PRS may actually give marginally better performance in maintaining consistency and state. The probability that the largest fingers are dead is actually lower for PRS then PNS, though only very slightly.

Of course these results will depend on the specific latency distribution used for the links in the network, as well as the specific PNS and PRS algorithms used. For Figs. 3 and 4, we have used the average delay per link to be 19 times faster than the session time of nodes. For PRS and PNS, we also need to specify what fraction of the links are good or bad. We assume that 0.01 percent of the links are 20 times slower. If that has still to give an average delay per link to be 19 times faster than the average session time of nodes, then the time taken by the faster links to transmit is about 19.2 times the average session time for nodes, while the time taken by the slower links to transmit is only about 0.96 times the average session time.

We expect the difference between the PNS and PRS algorithm to increase, with an increase in the fraction of “bad” links, even if these bad links are not quite as bad as what we have assumed above.

Preliminary results show that the PRS scheme is actually more robust in the sense that the region of instability (the region of parameter space in which the larger fingers are almost always dead) is smaller, all other parameters being exactly the same.

Preliminary results also show that the difference between PRS and PNS can increase with the number of nodes in the system. Our investigations along these lines are under way.

7 Proximity neigbour selection (PNS)

In this section, we study a scheme for PNS in the context of Chord and show how we can analyse it within our framework.

We assume that every node tries to only have fingers which transmit at a fast rate. This could imply that a node has *no* fingers in an interval. But this will happen only for the smaller intervals and does not substantially affect anything.

We assume that each node chooses its fingers in the following way. To get a contact in section $2^{i-1} \leq m < 2^i$ (its i^{th} finger), the node intiates a lookup for id 2^{i-1} in the usual way. With a probability $1-p$ the first successor of this is a good link in which case this is kept as the i^{th} finger. With a probability p however, it is a bad link. In this case, the node pings all the successors of this node consecutively (we assume that for every lookup for a finger, the target node also sends back its full successor list) till it finds a good link. In practice as shown in some studies, a node will with high probability not need to ping more than 10 nodes, before it finds a suitable finger candidate.

A few properties of this scheme that are useful are mentioned below.

The conditional probability of having the first node in the interval as a finger entry *given* that there exists at least one node in the interval is $(1-p)$.

Hence the simultaneous probability of having the first node in the interval as a finger entry *and* there being at-least-one node in the interval is $(1-p)a(m)$. $a(m)$ is derived from the geometric distribution of nodes on the ring as mentioned earlier.

To be more accurate, these probabilities also depend on churn. However, we ignore this dependancy for the moment.

We need to recompute f_k for this system, as well as rewrite the lookup equation 8 to take into account the fact that now fingers in a sector, need not necessarily be the first node in that sector.

7.1 f_k

The term that we need to recompute is $c_{3,2}$ since the join and fail protocols are the same as before and give the same terms. $c_{3,2}$ is the probability that in a time Δt, a wrong k^{th} finger is corrected. This happens when a node with a dead k^{th} finger sends out a lookup for this finger (at rate $\frac{\lambda_s}{log_2 N}(1-\alpha)$). The lookup has a probability a^{c_k} of being successful as before and in addition takes some time which gives the factor $[\frac{\lambda_t}{\lambda_t+\lambda_s c_k}]$. All this is exactly as before. In addition the first successor of the key is accepted as a finger entry only if its a 'good' link with probability $1-p$.

Putting this together we get:

$$c_{3.2} = \frac{\lambda_s}{log_2(N)}(1-\alpha)[\frac{\lambda_t}{\lambda_t+\lambda_s c_k}](1-p)S_1 a^{c_k}(\Delta t) \tag{10}$$

+ a term quantifying the rate of finding a good link by pinging the successors.

The definition of a is as before, however we no longer mean average rate of decay by λ_t. λ_t is now the rate of transmission of a *good* link since now, with high probability, all the connections of a node are *good* links.

If we want to compare the result of this scheme, with the proximity unaware scheme, then we need to fix the parameters here accordingly. In the greedy lookup scheme we took $\frac{\bar{\lambda}_t}{\lambda_t+\lambda_f} = 0.95$ as the worst-case value.

This fixed the ratio $\frac{\bar{\lambda}_t}{\lambda_f} = \frac{0.95}{.05} = 19$. In our PNS scheme

$$\frac{\bar{\lambda}_t}{\lambda_f} = x\frac{\lambda_t}{\lambda_f}p + (1-p)\frac{\lambda_t}{\lambda_f} \tag{11}$$

If $x \approx 0.05$ and $p = 0.01$ then

$$[\frac{\lambda_t}{\lambda_f}] = \frac{\bar{\lambda}_t/\lambda_f}{xp+1-p} \approx 19.2 \tag{12}$$

We still need to understand the factor S_1 in the first term, as well as how to compute the second term.

To do this, assume that a node can be in one of 2 states. State s_1 is the default state that all nodes are in.

A node goes into the state s_2 when a normal lookup does not find it a good link. In this case, it has to ping all the successors one by one, till it finds a replacement for its finger. It is useful to make this difference because of the different actions that a node in these two states can take. In state s_1 a node can fail, initate a stabilisation action for its own successor or finger or transmit a message for another node.

However in state s_2, a node prioritises pinging the target nodes to find an appropriate entry for its finger table. So a node in state s_2 can either fail or contiune its search for a good link.

As usual, we use S_1 or S_2 to denote the number of nodes in these states. We make further distincions between nodes in state s_2 in the following manner. Define $S_2^{(1)}$ as the nodes yet to ping the first successor.

$S_2^{(2)}$ as the number of nodes yet to ping the second sucessor (because the first successor did not provide a good connection) and so on.

We can quantify the changes in $S_2^{(1)}$ in the following manner.

$$S_2^{(1)} \rightarrow \begin{cases} S_2^{(1)} + 1 & \frac{\lambda_s}{log_2 N}(1-\alpha)a^{c_k}p[\frac{\lambda_t}{\lambda_t+\lambda_t c_k}]S_1 \Delta t \\ S_2^{(1)} - 1 & \lambda_f \Delta t S_2^{(1)} \\ S_2^{(1)} - 1 & [\lambda_t(1-p)\Delta t + x\lambda_t p \Delta t]S_2^{(1)} \end{cases} \tag{13}$$

The first term is the same as appears in the expression for $c_{3,2}$ except that now we multiply with a p instead of $(1-p)$. This accounts for the case when an ordinary lookup gives a bad connection and the node now needs to start pinging its neigbours. The second term accounts for the case when a node in state $s_2^{(1)}$-type fails.

The 3^{rd} term accounts for the case when either an $S_2^{(1)}$ node finds a good connection [with probability $(1-p)\lambda_t$] in which case it sets its finger entry to this value and goes back to the s_1 state. Or it does not find a good connection with probability $[xp\lambda_t]$ and it goes into the $s_2^{(3)}$ state.

Similarly for $S_2^{(2)}$ we have

$$S_2^{(2)} \rightarrow \begin{cases} S_2^{(2)} + 1 & [x\lambda_t p \Delta t]S_2^{(1)} \\ S_2^{(2)} - 1 & [\lambda_f \Delta t S_2^{(2)}] \\ S_2^{(2)} - 1 & [\lambda_t(1-p)\Delta t + x\lambda_t p \Delta t]S_2^{(2)} \end{cases} \tag{14}$$

Out of all these equations we only need to know what is S_2 $(= S_2^{(1)} + S_2^{(2)} + S_2^{(3)} + ... = N - S_1)$. Since the term of interest , the second term in the expression for $c_{3,2}$, is $\lambda_t(1-p)S_2$. Solving for S_2 we get

$$S_2 = \frac{qS_1}{1-b} \tag{15}$$

where

$$b = \frac{x\lambda_t p}{\lambda_f + [x\lambda_t p + (1-p)\lambda_t]} < 1 \tag{16}$$

and

$$q = \frac{(1-\alpha)\frac{\lambda_s}{log_2(N)}a^{c_k}p[\frac{\lambda_t}{\lambda_t+\lambda_s c_k}]}{\lambda_f + [x\lambda_t p + (1-p)\lambda_t]} \tag{17}$$

Since $S_1 + S_2 = N$ we get

$$S_1 + \frac{q}{1-b}S_1 = N \Rightarrow S_1 = \frac{N}{[1 + q/(1-b)]} \tag{18}$$

Putting this in we finally get

$$c_{3,2} = [(1-\alpha)\frac{\lambda_s}{log_2(N)}a^{c_k}(1-p)\frac{\lambda_t}{\lambda_t + \lambda_s c_k}][\frac{1}{1+q/(1-b)}]+\lambda_t(1-p)[\frac{q/(1-b)}{1+q/(1-b)}] \tag{19}$$

As before we need to solve the equations for the f_k's simultaneously with the equations for the c_k's. However we need to rewrite the lookup equation (Eq. 8) slightly so that we account for the possibility that the first node in the interval need not always be the finger.

7.2 Lookup Equation for the PNS scheme

The lookup equation has the same logical structure as Eq. 8 except for one difference. The coefficient $a(m)$ - the probability of having at least one node in an interval of size m, is replaced by $y(m)$, the probability of having a finger in an interval m, since we are no longer guaranteed that the first node in an interval is the finger of choice.

$y(m)$ is easily calculated recursively in the following manner. Let $y^{'}(m)$ be the probability of *not* having a finger in an interval of size m. Clearly $y^{'}(m) = y^{'}(m-1)(\rho + (1-\rho)p)$, that is, the probability of not having a finger in an interval of size m is the same as the probability of not having a finger in an interval of size $m-1$, times the probability that either there exists no node at the m^{th} spot or there is one but with a bad link.

Solving the recursion, we get $y(m) = 1 - (\rho + (1-\rho)p)^m$. In the case that $p = 0$ when all links are the same, $y(m)$ is the same as $a(m)$.

With this, we can write the lookup equation for the PNS system as

$$\begin{aligned}
&C_{\xi+m} = C_\xi\,[1-y(m)]\,(1-p) \\
&+ [1-y(m)]\,p\Bigg[1+\sum_{i=1}^{k-1}h_k(i) \\
&\sum_{l=0}^{\xi/2^i-1} bd(l,\xi/2^i)(1+(i-1)+C_{\xi_i-l+m}) + O(h_k(k))\Bigg] \\
&+ (1-f_k)y(m)\left[1+\sum_{i=0}^{m-1} bd(i,m)C_{m-i}\right] \\
&+ f_k y(m)\Bigg[1+\sum_{i=1}^{k-1}h_k(i) \\
&\sum_{l=0}^{\xi/2^i-1} bd(l,\xi/2^i)(1+(i-1)+C_{\xi_i-l+m}) + O(h_k(k))\Bigg]
\end{aligned} \tag{20}$$

The logical structure is almost the same as before. The four terms enumerate the four possibilities. Either there is no finger in the interval but the first successor is still the finger (with probability $1-p$), there is no finger in the interval and the first successor is not the finger (with probability p), there is an alive finger in the interval or there is a dead finger in the interval. The function h_k are the same as before and the functions bd's are the counterpart of the bc's which appear in Eq. 8 and are given in terms of the $y(m)$'s as

$$bd(i,m) = \frac{(1-y(i-1))(1-\rho)(1-p)}{y(m)}$$

As before we solve simultaneously for the f_k's and the lookup cost, to get the plots in Figs 3 and 4.

8 Proximity Route Selection

We briefly demonstrate here how we can also analyse proximity route selection. In this case we do not need to recompute the probability of fingers being dead. This is the same as equation as Eq. 7. But we need to consider a PRS scheme for routing. There clearly exist many such schemes [20], but we consider the simplest of them which is one based on pure proximity. In this case, a node, uses a finger for routing *only* if its a good link, no matter how far back it has to hop from the target. Since the manner in which fingers are chosen is the same as in the case without proximity, The lookup equation is the same as Eq. 8 except that the h_k's are now a little different, to account for the fact that a jump back to an earlier finger can be made even if the finger was alive but a bad link.

The expression for the h_k's is hence:

$$\begin{aligned} h_k(i) =& a(\xi/2^i)(1-f_{k-i})(1-p) \\ & \times \Pi_{s=1,i-1}(1-a(\xi/2^s)+a(\xi/2^s)f_{k-s}+(1-a(\xi/2^s))(1-f_{k-s})p), i<k \\ h_k(k) =& \Pi_{s=1,k-1}(1-a(\xi/2^s)+a(\xi/2^s)f_{k-s}+(1-a(\xi/2^s))(1-f_{k-s})p) \end{aligned} \tag{21}$$

Again we solve the lookup equation and the equation for the f_k's simultaneously to get the results in Figs 3 and 4.

9 Discussion and Conclusion

In this paper, we have presented an analytical framework for analysing proximity related issues in P2P networks. Our methods, though carried out for Chord, can be carried over to other systems. In addition, the analysis has a naturally modular structure, which makes it easy to redo the analysis for different design choices or parameters that go into building a DHT.

Our analysis predicts that there is a region in parameter space where the ring, though still connected, does not carry out efficient searches because of the longest fingers always being dead. In addition we are also able to compare the PNS and PRS schemes for varying system sizes and parameters.

For the future, we also intend to think about congestion in this framework. That is, we will not only think of messages being passed over links but also how big these messages are, and how frequent. This could affect the average delay in a network, leading to interesting consequences.

10 Acknowledgements

We would like to thank Ali Ghodsi for several useful discussions. SK is funded by the swedish research council.

References

[1] James Aspnes, Zoë Diamadi, and Gauri Shah, *Fault-tolerant routing in peer-to-peer systems*, Proceedings of the twenty-first annual symposium on Principles of distributed computing, ACM Press, 2002, pp. 223–232.

[2] Miguel Castro, Manuel Costa, and Antony Rowstron, *Performance and dependability of structured peer-to-peer overlays*, Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04), IEEE Computer Society, 2004.

[3] F. Dabek, J. Li, E. Sit, J. Robertson, M.F. Kaashoek, and R. Morris, *Designing a DHT for low latency and high throughput*, Proc. NSDI **4** (2004).

[4] A. Datta and K. Aberer, *Internet-scale storage systems under churn2̆014A study of the steady-state using Markov models*, Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing (2006), 133–144.

[5] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, *The impact of DHT routing geometry on resilience and proximity*, Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (2003), 381–394.

[6] O. Herrera and T. Znati, *Modeling Churn in P2P Networks*, Proceedings of the 40th Annual Simulation Symposium (ANSS'07)-Volume 00 (2007), 33–40.

[7] C. Jin, H. Wang, and K.G. Shin, *Hop-count filtering: an effective defense against spoofed DDoS traffic*, Proceedings of the 10th ACM conference on Computer and communications security (2003), 30–41.

[8] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, *Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web*, Proceedings of the twenty-ninth annual ACM symposium on Theory of computing (1997), 654–663.

[9] F. Klemm, J.Y. Le Boudec, D. Kostic, and K. Aberer, *Improving the Throughput of Distributed Hash Tables Using Congestion-Aware Routing*.

[10] S. Krishnamurthy, S. El-Ansary, E. Aurell, and S. Haridi, *A statistical theory of chord under churn*, Proceedings of the 4th IPTPS (2005).

[11] Supriya Krishnamurthy, Sameh El-Ansary, Erik Aurell, and Seif Haridi, *A statistical theory of chord under churn*, The 4th International Workshop on Peer-to-Peer Systems (IPTPS'05) (Ithaca, New York), February 2005.

[12] ———, *An analytical study of a strutured overlay in the presence of dynamic membership*, IEEE Joint Transactions on Networking, in press (2007).

[13] ———, *Comparing maintenance strategies for overlays*, Tech. report, Swedish Institute of Computer Science, Proceeedings of PDP2008 2007.

[14] J. Li, J. Stribling, T.M. Gil, R. Morris, and M.P. Kaashoek, *Comparing the Performance of Distributed Hash Tables Under Churn*, Peer-To-Peer Systems III: Third International Workshop, IPTPS 2004, La Jolla, CA, USA, February 26-27, 2004: Revised Selected Papers (2004).

[15] Jinyang Li, Jeremy Stribling, Robert Morris, M. Frans Kaashoek, and Thomer M. Gil, *A performance vs. cost framework for evaluating dht design tradeoffs under churn*, Proceedings of the 24th Infocom (Miami, FL), March 2005.

[16] David Liben-Nowell, Hari Balakrishnan, and David Karger, *Analysis of the evolution of peer-to-peer systems*, ACM Conf. on Principles of Distributed Computing (PODC) (Monterey, CA), July 2002.

[17] B.T. Loo, T. Condie, J.M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica, *Implementing declarative overlays*, ACM SIGOPS Operating Systems Review **39** (2005), no. 5, 75–90.

[18] D. Malkhi, M. Naor, and D. Ratajczak, *Viceroy: A scalable and dynamic emulation of the butterfly*, Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC '02), August 2002.

[19] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, *A Scalable Content Addressable Network*, Proceedings of the ACM SIGCOMM '01 Conference (Berkeley, CA), August 2001.

[20] S. Rhea, B.G. Chun, J. Kubiatowicz, and S. Shenker, *Fixing the embarrassing slowness of OpenDHT on PlanetLab*, Proc. WORLDS (2005).

[21] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz, *Handling churn in a DHT*, Proceedings of the 2004 USENIX Annual Technical Conference(USENIX '04) (Boston, Massachusetts, USA), June 2004.

[22] Antony Rowstron and Peter Druschel, *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*, IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 329-350 2001.

[23] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan, *Chord: A scalable peer-to-peer lookup service for internet applications*, IEEE Transactions on Networking **11** (2003).

[24] Shengquan Wang, Dong Xuan, and Wei Zhao, *On resilience of structured peer-to-peer systems*, GLOBECOM 2003 - IEEE Global Telecommunications Conference, Dec 2003, pp. 3851–3856.

[25] Ben Y. Zhao, Ling Huang, Sean C. Rhea, Jeremy Stribling, Anthony D Joseph, and John D. Kubiatowicz, *Tapestry: A global-scale overlay for rapid service deployment*, IEEE J-SAC **22** (2004), no. 1, 41–53.