Abstract. In this paper, we present an analytical tool for understanding the performance of structured overlay networks under churn based on the master-equation approach of physics. We motivate and derive an equation for the average number of hops taken by lookups during churn, for the Chord network. We analyse this equation in detail to understand the behaviour with and without churn. We then use this understanding to predict how lookups will scale for varying peer population as well as varying the sizes of the routing tables. We also consider a change in the maintenance algorithm of the overlay, from periodic stabilisation to a reactive one which corrects fingers only when a change is detected. We generalise our earlier analysis to understand how the reactive strategy compares with the periodic one.

Keywords: Peer-To-Peer, Structured Overlays, Distributed Hash Tables, Dynamic Membership in Large- scale Distributed Systems, Analytical Modeling, Master Equations.

---

1This work is funded by the 6th FP EVERGROW project.
Comparing Maintenance Strategies for Overlays

Supriya Krishnamurthy\(^1,3\), Sameh El-Ansary\(^1\), Erik Aurell\(^1,2,4\) and Seif Haridi\(^1,3\)
\(^1\) Swedish Institute of Computer Science (SICS), Sweden
\(^2\) Department of Computational Biology, KTH-Royal Institute of Technology, Sweden
\(^3\) Department of Information and Communication Technology, KTH-Royal Institute of Technology, Sweden
\(^4\) ACCESS Linnaeus Center, KTH- Royal Institute of Technology, Sweden
{supriya,sameh,eaurell,seif}@sics.se

Abstract—In this paper, we present an analytical tool for understanding the performance of structured overlay networks under churn based on the master-equation approach of physics. We motivate and derive an equation for the average number of hops taken by lookups during churn, for the Chord network. We analyse this equation in detail to understand the behaviour with and without churn. We then use this understanding to predict how lookups will scale for varying peer population as well as varying the sizes of the routing tables. We also consider a change in the maintenance algorithm of the overlay, from periodic stabilisation to a reactive one which corrects fingers only when a change is detected. We generalise our earlier analysis to understand how the reactive strategy compares with the periodic one.

I. INTRODUCTION

A crucial part of assessing the performance of a structured P2P system (aka DHT) is evaluating how it copes with churn. Extensive simulation is currently the prevalent tool for gaining such knowledge. Examples include the work of Li et al. [10], Rhea et al. [13], and Rowstron et al. [5]. There has also been some theoretical analyses done, albeit less frequently. For instance, Liben-Nowell et al. [11] 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. [4] 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.

We have chosen to adopt a slightly different approach to theoretical work on DHTs. We concentrate not on establishing bounds, but rather on a more precise prediction of the relevant quantities in such dynamically evolving systems. Our approach is based mainly on the Master-Equation approach used in the analysis of physical systems. We have previously introduced our approach in in [7], [8] where we presented a detailed analysis of the Chord system [14]. In this paper, we show that the approach is applicable to other systems as well. We do this by comparing the periodic stabilization maintenance technique of Chord with the correction-on-change maintenance technique of DKS [3].

Due to space limitations, we assume reader familiarity with Chord and DKS, including such terminology as successors, finger starts and finger nodes etc.

The rest of the paper is organised as follows. In Section II, we introduce the Master-Equation approach. In Section III, we mention some related work. In Section IV we begin by briefly reviewing some of our previously published results on predicting the performance of the Chord network as a function of the failed pointers in the system in the case that the nodes use a periodic maintenance scheme. We then show some new results on how this complicated equation can be simplified to get quick predictions for varying number of peers and varying number of links per node. We relegate some of the details of this analysis to Appendix VII. In Section V, we explain how to use the Master-Equation approach to analyse the reactive maintenance strategy of interest and present our results on how this strategy compares with the periodic case analysed earlier. We summarise our results in Section VI.

II. THE MASTER-EQUATION APPROACH FOR STRUCTURED OVERLAYS

In a complicated system like a P2P network, in which there are many participants, and in which there are many interleaved processes happening in time, predicting the state of the network (or of any quantity of interest) can at best be done by specifying the probability distribution function (PDF) of the quantity in the steady state (when the system, though changing continually in time, is stationary on average). For example, one quantity which affects the performance of the network and hence of interest to us, is the fraction of failed links between nodes, in the steady state. The problem is thus to calculate the PDF (or the average in the steady state) of this quantity and then to understand quantitatively, how it affects the performance of the network.

In general this is not an easy task, since the probability is affected by a number of inter-leaved processes in any time-varying system. In [7], [8], we demonstrated how we could analyse a P2P network like Chord [14], using a Master-Equation based approach. This approach is generally used in physics to understand a system evolving in time, by means of equations specifying the time-evolution of the probabilities of finding the system in a specific state. In the context of a P2P network, the state of the system could be specified by, how many nodes there are in the network and what the state (whether correct, incorrect or failed) of each of the pointers of those nodes is. The equations for the time-evolution of the system then require as an input, the rates of various processes...
affecting the state of the system. These processes should ideally be independent of each other, so that they entirely determine the time-evolution of the network. For example, in a peer-to-peer network, these processes could be the join and failure rates of the member nodes, the rate at which each node performs maintenance as well as the rate at which lookups are done in the network (the latter rate is relevant only if the lookups affect the state of the network in some way). Given these rates, the equation for the time-evolution of the probability of the quantity of interest can be written by keeping track of how these rates affect this quantity (such as the number of failed pointers in the system) in an infinitesimal interval of time, when only a limited number of processes (typically one) can be expected to occur simultaneously.

With this approach, we were able to quantify very accurately the probabilities of any connection in the network (either fingers or successors) having failed. We then demonstrated how we could use this information to predict the performance of the network—the number of hops including time outs which a lookup takes on average — as a function of the rates (of join, failure and stabilization) of all the processes happening in the network, as well as of all the parameters specifying the network (such as how many pointers a node has on average). The analysis was done for a specific maintenance strategy, called periodic maintenance (or eager maintenance).

In this paper, we generalise our approach so as to be able to compare networks using different maintenance strategies. In particular, we compare our earlier results for periodic maintenance with a reactive maintenance strategy proposed in [6]. Combining this with some of our previous results, we are also, as a by product, able to compare the performance of networks specified by different numbers of peers, different number of pointers per node and/or different maintenance strategies. As we show below, which system is better depends both on the value of the parameters as well as the level of churn. The approach we propose is thus a useful tool for the quantitative and fair comparison of networks specified by different parameters and using different algorithms.

III. RELATED WORK

In [2], an analysis, very similar in spirit to the one done in this paper, is carried out in the context of P-Grid [1]. An equation is written for system performance in the state of dynamic equilibrium for various maintenance strategies. However for each maintenance strategy, the analysis has to be entirely redone. In contrast, a master equation description [12] provides a foundation for the theoretical analysis of overlays, which does not have to be entirely rebuilt each time any given algorithm is changed. As we show in this paper, we can carry over a lot of our earlier analysis, when the maintenance scheme is changed from a periodic to a reactive one. In addition, the master equation description can be made arbitrarily precise to include non-linear effects as well. And as we show, non linear effects are important when churn is high.

IV. THE LOOKUP EQUATION FOR CHORD

We quantify the performance of the network, by the number of hops required on average from the originator of the query to the node with the answer. This is just the total number of nodes contacted per query (or equivalently, the total number of pointers used per query) including the total number of failed pointers used en route. This latter quantity (which arises because of the churn in the network) is the reason that the hop count per query increases with high dynamism and is hence an important quantity to understand. In the case of the periodic maintenance scheme, this quantity is a function of \((1 - \beta) r\) where \(r\) is the ratio of the stabilisation rate to the join (or failure) rate and \(1 - \beta\) is the fraction of times a node stabilises its finger, when performing maintenance, as mentioned in Section I. We demonstrate how this quantity can be calculated in Section V, in the context of the reactive maintenance policy, which is a simple generalisation of how it is calculated earlier in [7], [8], for the periodic maintenance scheme. In this section, we briefly review our earlier results on how the performance of the network (as exemplified by the average hopcount per query), can be determined once the fraction of failed pointers is known.

The key to predicting the performance of the network is to write a recursive equation for the expected cost \(C(t, \beta)\) (also denoted \(C(t)\)) for a given node to reach some target, \(t\) keys away from it. (For example, \(C(t)\) is the cost of looking up the adjacent key which is \(1\) key away).

The Lookup Equation for the expected cost of reaching a general distance \(t\) is then derived by following closely the Chord protocol which is a greedy strategy designed to reduce the distance to the query at every step without overshooting the target. A lookup for \(t\) thus proceeds by first finding the closest preceding finger. The node that this finger points to is then asked to continue the query, if it is alive. If this node is dead, the originator of the query uses the next closest preceding finger and the query proceeds in this manner.

For the purposes of the analysis, it is easier to think in terms of the closest preceding start. Let us hence define \(\xi\) to be the start of the finger (say the \(k^{th}\)) that most closely precedes \(t\). Hence \(\xi = 2k - 1 + n\) and \(t = \xi + m\), i.e. there are \(m\) keys between the sought target \(t\) and the start of the most closely preceding finger. With that, we can write a recursion relation for \(C_{\xi + m}\) as follows:

\[
C_{\xi + m} = C\xi \left[1 - a(m)\right] + (1 - f_k) a(m) \left[1 + \sum_{i=0}^{m-1} bc(i, m) C_{m-i}\right] + f_k a(m) \left[1 + \sum_{i=1}^{k-1} h_k(i) \xi/2^l - 1 \sum_{l=0}^{bc(l, \xi/2^l)(1 + (i - 1) + C_{\xi, -l+m}) + O(h_k(k))} \right]
\]

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 \(a, bc\) can be derived from the internode interval distribution [7], [8] which is just the distribution of distances between adjacent nodes. Given a ring of \(K\) 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 $\rho = \frac{\mu}{\lambda}$. 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 $\rho^i(1-\rho)$. 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 $h_k(i)$ is easy to compute given the probability $a$ as well as the probabilities $f_k$’s of the $k^{th}$ finger being dead.

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

$$h_k(k) = \Pi_{s=1,k-1}(1-a(\xi/2^s) + a(\xi/2^s)f_{k-s})$$

Eqn.2 accounts 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 (fingers $k$ and $k-1$ are not distinct if they have the same entry in the finger table. Though the $starts$ of the two fingers are different, if there is no node in the interval between the $starts$, the entry in the finger table will be the same). The probabilities $h_k(i)$ satisfy the constraint $\sum_{i=1}^{k} h_k(i) = 1$. $h_k(k)$, is the probability that a node cannot use any earlier entry in its finger table, in which case it has to fall back on its successor list instead. We indicate this case by the last term in Eq. 1 which is $O(h_k(k))$. In practise, 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 for the moment.

The cost for general lookups is

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

The lookup equation is solved recursively numerically, using the expressions for $a$, $bc$, $h_k(i)$ and $C_1$. In Fig. 1, we have plotted the theoretical prediction of Equation 1 versus what we get from simulating Chord. Here we have used $N \approx 1000$ and $K = 2^{10}$. To get an idea of what the parameter $\beta$ means, we take an example of some values taken from an actual model is slightly different because we take an example of some values taken from an actual system, finger stabilisation intervals are in the range between 40 seconds and 19 minutes and successor stabilisation rates are in the range between 4 seconds and 19 minutes. While our model is slightly different because we either stabilise fingers or successors while Li et al (10) do both independently, nevertheless, we can roughly translate their values to imply a $(1-\beta)r$ lying between 90 and 3, while $r$ lies between 990 and 6. In our simulations, the lowest $r$ value we were able to achieve was $\sim 25$. This is because we did not take into account some optimisations in the Chord protocol [14] such as using lookups (which are assumed to take place every 10 minutes in [10]) to correct wrong information. This could increase the effective $1-\beta$ value in [10]. Another obvious optimisation which we have not used is the fact that not all of the fingers are distinct and many of the smaller fingers are actually the same as successors and hence can be corrected from the information obtained using successor stabilisations.

If we take all these factors into account, then the parameter values we have looked at should be effectively similar to the ones studied in actual implementations.

As can be seen the theoretical results match the simulation results very well. In Fig. 2 we also show the theoretical predictions for some larger values of $N$.

On general grounds, it is easy to argue from the structure of Equation 1, that the dependence of the average lookup on churn comes entirely from the presence of the terms $f_k$. Since $f_k \sim f$ is independent of $k$ for large fingers, we can approximate the average lookup length by the functional form $L(r, \beta) = A + Bf + Cf^2 + \cdots$. The coefficients $A, B, C$ etc can be recursively computed by solving the lookup equation to the required order in $f$. They depend only on $N$ the number of nodes, $1-\rho$ the density of peers and $b$ the base or equivalently the size of the finger table of each node. The advantage of writing the lookup length this way is that churn-specific details such as how new joinees construct a finger table or how exactly stabilizations are done in the system, can be isolated.
in the expression for $f$. If we were to change our stabilization strategy, as we will demonstrate below, we could immediately estimate the lookup length by plugging in the new expression for $f$ in the above relation.

Another advantage of having a simple expression such as the above, is that if we can estimate $A$, $B$, $C$ · · · accurately, we can make use of the expression for $L$ to estimate the churn (or the value of $r$) in the system, hence using a local measure to estimate a global quantity. The logic in doing so is the inverse of the reasoning we have used so far. So far, we have used the churn as the input for finding $f_k$ and hence $L$. But we can also reverse the logic and try and estimate churn, if we know the value of the average lookup length $L$. If $L$ has the above simple expression, then given $A$ and $B$ to $O(f)$, we have $f = \frac{L-A}{B}$. From the expression for $f$ (see Section V for how to evaluate $f$), we can now get the value of $r$. Hence any peer can make an estimate of the churn that the system is facing if it knows how long its lookups are taking on average, and if it has an estimate of $N$.

To get $A$, we need to consider Eqn 1 with no churn (all $f_k$’s set to zero). In Appendix VII, we study the lookup equation 1 in some detail to understand the behaviour without churn and obtain the value of $A$ for any base $b$. This is useful on several counts. First, the value of $A$ is needed to predict the lookup costs as explained above. Secondly, if $b$ changes (a system of base $b$ has a finger table of size $M = (b-1)\log_b(K)$), all else remaining the same, the only major change in the lookup cost is due to the change in $A$. So estimating $A$ precisely has the benefit that we can predict the lookup cost for any base $b$. Thirdly, the analysis confirms that Equation 1 does indeed reproduce well known results for the lookup hop count in Chord, such as for example, that the average lookup cost is $0.5 \times \log(N)$ without churn [14]. Instead as demonstrated in Appendix VII, for any $N$, the average lookup cost as predicted by Eq. 1 is indeed $0.5 \times \log(N)$ plus some $\rho$-dependent corrections which though small are accurately predicted.

A simple estimate for $B$ and $C$ can be made in the following manner. Let every finger be dead with some finite probability $f$. Each lookup encounters on average $A$ fingers, where $A$ is the average lookup length without churn. Each of these fingers could be alive (in which case it contributes a cost of 1), dead with a probability $f$ in which case it contributes a cost of 2 if the next finger chosen is alive (with probability $1-f$) and so on. It is trivial to verify that this estimates the look-up cost to be $A(1+f+f^2+\cdots)$. Comparing with our expression for $L$, this gives an estimate of $B = A, C = A, \cdots$.

In general if $L = A+B \times g(f)$, then if we scale $L$ by plotting $(L-A)/B$ for varying $N$, we should get an estimate of $g(f)$. Note that $f$ depends on $\rho$ and $M$ the number of fingers. In addition if $g(f) = a_1 f + a_2 f^2 + \cdots$, the coefficients $a_1, a_2, \text{etc}$ can also depend on $\rho$. However for $1 - \rho << 1$, these dependences on $\rho$ are small and the curves for different $N$ collapse onto the same curve on scaling. In Fig. 3 we have scaled the curves plotted in Fig. 2 in the above manner, using $B = A$. The values of $A$ used are derived from the analysis of the previous section. As can be seen the curves collapse onto one curve which is well approximated by the function $g(f) = f + 3 - f^2$, giving $a_1 = 1$ and $a_2 = 3$. The fits in Fig. 2 are also according to this functional form. It should be emphasized however that this approximation for $g(f)$ is good only for $1 - \rho << 1$. For higher values of peer density, the curves for different $N$ will not collapse onto one curve and any $\rho$-dependence of the coefficients $a_i$’s will show up as well.

We can use the above functional form to predict how lookups would behave if we change the base $b$ (the size of the routing table) of the system. In Fig. 4 we plot the functional form $A(b)(1+f(b)+3f(b)^2)^2$ for $b = 2, 4, 16$. The coefficient $A(b)$ is accurately predicted by Eq. 11 (in Appendix VII), with the definition of $\xi(i+1)$ taken appropriately. $f(b)$ is affected by the base $b$ because the number of fingers increases with $b$.

As can be seen, when churn is low, a large $b$ is an advantage and significantly improves the lookup length. However when churn is high, the flip side of having a larger routing table is that it needs more maintenance. Hence beyond some value of churn, the larger the value of $b$, the larger the lookup latency.

This is similar to the spirit of the numerical investigations done in [10]. However when comparing different bases for Chord, Li et al [10] find that while base 2 is the best for high churn (as we find here), base 8 is the best for low churn. Increasing the base beyond this does not seem to improve the cost. The discrepancy between this finding and ours is due to the details of the periodic maintenance scheme which we use. In our case, we have taken the simplest scenario in which each node needs to stabilise $M$ fingers and the order in which this is done is random. In practice only $\sim \log N$ of the $M$ fingers are distinct, so only $\sim \log N$ stabilisations need be done by each node. In addition, in [10], finger stabilisations are done only if the finger is pinged and found to be dead.

V. ‘Correction-on-Change’ Maintenance Strategy

In this section, we analyse a different maintenance strategy using the master-equation formalism. The strategy we have analysed so far is periodic stabilisation of successors as well as fingers. We now consider a strategy where a node periodically stabilises its successors but does not do so for its fingers. Instead, for maintaining its fingers, it relies on other nodes for updates [6]. Whenever a node $n$ detects that its first successor
comparison meaningful, we need to quantify the concept of 'maintenance-effort' per node, and compare the two schemes at a given level of churn and at the same value of the maintenance effort per node. We elaborate on this a little later in Section V-B.

Another point to note is how to quantify system performance. We have previously done it in terms of lookup hops. But a more correct way might be to ask for the latency for consistent lookups (since some of the lookups could be inconsistent). However we have checked that, within our analytical framework, this does not change the results qualitatively.

A. Analysis of the Correction-on-change strategy

To generalise the analysis to meet the situation when some nodes are sending messages while others are not, we say that a node can be in state $S_1$ or $S_2$. In state $S_1$, a node can stabilise its first successor at rate $\alpha \lambda$, fail at rate $\lambda f$ and assist in joins at rate $\lambda j$ as before. In state $S_2$, a node can stabilise its first successor at rate $\alpha \lambda$, fail at rate $\lambda f$, assist in joins at rate $\lambda j$ and in addition, send correction messages (which is essentially equivalent to doing one lookup) at rate $\lambda M \equiv c \lambda$. As we show in Section V-B, if we want to compare the two maintenance strategies in a fair manner then the most general values that these parameters can take is $\alpha = 1$ and $\alpha + c = 1$.

Let $N_{S_1}$ be the number of nodes in state $S_1$ and $N_{S_2}$ the number of nodes in state $S_2$. Clearly $N_{S_1} + N_{S_2} = N$, the total number of nodes in the system.

We can further partition $S_2$ into $S_2^1, S_2^2, S_2^3, \cdots, S_2^{M}$. $S_2^1$ is the state of the node which has yet to send its first correction message, $S_2^2$ the state of the node which has sent its first correction message but is yet to send its second, etc.

Consider the gain and loss terms for $N_{S_1}$. These are summarised in Table I.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_{1.1}$</td>
<td>The probability that an $S_1$ node is lost because it failed.</td>
</tr>
<tr>
<td>$c_{1.2}$</td>
<td>The probability that a join occurs thus adding to the number of $S_1$ nodes in the system (since a new joinee is always an $S_1$-type node).</td>
</tr>
<tr>
<td>$c_{1.3}$</td>
<td>The probability that an $S_2^M$ node sent its last message at rate $\lambda_M$ and converted into an $S_1$ node. The last term $c_{1.4}$ is the probability that an $S_1$-type node did a stabilisation at rate $\alpha \lambda$, found a wrong first successor with probability $w_1$ and hence converted into an $S_2$ node. $w_1$ is the fraction of wrong successor pointers of an $S_1$-type node.</td>
</tr>
</tbody>
</table>

Defining $\lambda_s/\lambda_f = r$ and $\lambda_M/\lambda_f = cr$ the steady state equation predicted by Table I is:

$$P_{S_1}(1 + crw_1) = 1 + crP_{S_2^M}$$

(3)

where $P_{S_1} = N_{S_1}/N$. 

**Table I**

<table>
<thead>
<tr>
<th>Gain and Loss Terms for $N_{S_1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{S_1}(t + \Delta t)$</td>
</tr>
<tr>
<td>$= N_{S_1}(t) + 1$</td>
</tr>
<tr>
<td>$= N_{S_1}(t) + 1$</td>
</tr>
<tr>
<td>$= N_{S_1}(t) + 1$</td>
</tr>
<tr>
<td>$= (\alpha \lambda N_{S_1}(t))w_1$</td>
</tr>
<tr>
<td>$= N_{S_1}(t)$</td>
</tr>
</tbody>
</table>

For a system with $f$ fingers pointing to any node (there can be more than one node-space), we argue below, for our purpose this is not important). Of course, this if node-spacings are smaller than average. However, as we take into account the probability that a correction action leads to new knowledge (since the successor stabilises at the first successor updated (because for example, the successors of $p$ also happen to point $n.s_1$), $n$ could leave the responsibility of informing these other nodes to $p$. We could take into account the probability that a correction action leads to more than $M$ messages. But for the moment we ignore this point (We could argue that once it is $p$’s responsibility to check that its successors know about $n.s_1$, it could piggy-back this information when it does a successor stabilisation, which does not affect the number of messages sent).

Whenever a node receives a message updating its information about a finger, it immediately corrects the appropriate entry in its routing table.

In the following, we demonstrate how we can analyse such a strategy. We would like to ultimately compare its performance to periodic stabilisation in the face of churn. To make such a
We can write a similar equation $N_{S_2}$ which however does not give us any new information since $N_{S_1} + N_{S_2} = N$.

Writing a gain-loss equation for each of the $N_{S_2}$'s in turn, we obtain,

$$P_{S_2} = \frac{P_{S_1} (arw_1 - arw'_1)}{1 + cr + arw'_1} + \frac{arw'_1}{1 + cr + arw'_1}$$

(4)

and

$$P_{S_2} = P_{S_2} \left( \frac{cr}{1 + cr + arw'_1} \right)^{i-1}$$

(5)

for $2 \leq i \leq M$.

Here $w_1$ is the fraction of $S_1$ nodes with wrong pointers and $w'_1$ is the fraction of $S_2$ nodes with wrong pointers. We have made a simplification here in assuming that the fraction of wrong pointers of $S_2$ nodes is the same, irrespective of the state of the $S_2$ node. In practice (especially if $a = 0$), this will not be the case. However for the parameter ranges we are interested in $(r > 1)$, this is not crucial.

Clearly $\sum_{i=1}^{M} P_{S_2} = P_{S_1}$. A quantity of interest in our analysis is

$$P_{S_2}/P_{S_1} = 1 - \frac{(1 - \frac{g_1^M-1}{g_1^M})}{1 - \frac{g_1^M}{g_1^M}}$$

(6)

where $g_1 = \frac{cr}{1 + cr + arw'_1}$.

To solve for $P_{S_1}$, etc., we need to solve for $w_1$ and $w'_1$.

However, consider first the equation for $W_T$ - the total number of wrong successor pointers in the system (irrespective of whether the pointer belongs to an $S_1$ or an $S_2$ type node. The gain and loss terms for $W_T$ are shown in table II. $w = W_T/N$ is the fraction of wrong successor pointers in the system.

This gives the following equation

$$(3 + ar)w_1P_{S_1} + (3 + ar)w'_1P_{S_2} = 2$$

(7)

The gain and loss terms $W_T$ - the number of $S_2$ nodes with wrong successor pointers - are written in much the same way except for a few small changes. Table III details the changes that occur in $W_T$ in time $\Delta t$.

The terms here are much the same as derived earlier except that we now have to keep track of whether the node that is failing in terms $c_{3,2}$ and $c_{3,3}$ is an $S_1$ or an $S_2$-type node. In addition term $c_{3,5}$ is the probability that a $S_2^{M}$-type node has a wrong successor pointer, but sends a message and hence turns into an $S_1$ node with a wrong pointer.

Table III gives us the following equation for $w'_1$ in the steady state.
An estimate for $\sum f_k$ is simply $\sim MN_{S_2}/N$. Substituting this in term $c_{4,2}$, this term becomes $= \lambda_M N \Delta t(f_k/M)(1 - w'_r)A(w_1, w'_1)$.

Solving for $f_k$ in the steady state, and substituting for $w'_r$, we get $f_k$ as a function of the parameters. As mentioned earlier a quick and precise estimate of the lookup length is then obtained by taking $L = A(1 + f + 3f^2)$.

**B. Comparison of Correction-on-change and Periodic Stabilisation**

In order to compare how the two strategies perform under churn, we need to make sure that we are comparing lookup latencies for the same number of total maintenance messages sent.

Let us assume that the maximum rate for sending messages per node is $C$. In the case of periodic stabilisation, this implies that the rate of doing successor stabilisations $\lambda_{S_1}$ and finger stabilisations $\lambda_{S_2}$ must in total not exceed $C$. This implies that $\lambda_{S_1}/C + \lambda_{S_2}/C \leq 1$. If we assume that all nodes always send messages up to their maximum capacity, then clearly $\lambda_{S_1}/C + \lambda_{S_2}/C = 1$. Suppose we define $r \equiv C/\lambda_j$ and $r_1 \equiv \lambda_{S_1}/\lambda_j$, $r_2 \equiv \lambda_{S_2}/\lambda_j$. Then for a given value of $r$, $r_1 + r_2 = r$.

Hence if finger stabilisations are done at rate $(1 - \beta)r$, the successor stabilisations need to be done at rate $\beta r$, where the parameter $\beta$ can be varied from 0 to 1.

In the case of correction-on-change, we need to impose the same maximum rate $C$ no matter which state the nodes are in. In this case, let $\lambda_{S_1}$ be the rate of successor stabilisation in state $S_1$, $\lambda_{S_2}$ the rate of successor stabilisation in state $S_2$ and $\lambda_{S_3}$ be the rate of sending messages in state $S_2$. Clearly $\lambda_{S_1} = C$ and $\lambda_{S_2} + \lambda_{S_3} = C$. Defining $r$ as before, we get $\lambda_{S_1}/\lambda_j = r$ and $\lambda_{S_2}/\lambda_j + \lambda_{S_3}/\lambda_j = r$. Hence comparing with our parameters $\alpha = 1$ and $a + c = 1$.

In Fig. 5, we have plotted the function $L = A(1 + f + 3f^2)$ with the value of the lookup length without churn $A = 5.846$ for $N = 1000$ nodes, for $a = 0$ (and $c = 1$) and for $\beta = 0.4$. $f$ is calculated separately for the two maintenance techniques.

As can be seen, correction-on-change is better than periodic stabilisation when churn is low but not when churn is high. On comparing lookup lengths for several different $a$, it becomes evident (see Fig. 6) that $a \sim 0.2$ is the optimum value for the correction-on-change strategy.

So interestingly, for nodes in state $S_2$, it is not the best strategy to increase $c$ as much as possible. It is a better strategy to spend some of the bandwidth on maintaining a correct successor.

To understand these results better, let us again translate the parameters, $a$, $c$ and $\alpha$ into numbers used in implementations. As we saw, the implementation of Chord in [10] has a value of $r$ ranging from $\sim 10$ to 1000. Take a representative $r$ value of 100. This implies that for an average session time of 1 hour, a stabilisation process (either successor or finger) takes place on average every 36 seconds. Hence in Fig. 5, we have compared two systems, one in which successor stabilisations happen every 40 seconds on average and finger stabilisations happen every 60 seconds on average. In the other system, $S_1$-type nodes stabilise successors every 36 seconds, and $S_2$-type nodes send messages every 36 seconds till they have sent $\sim M$ messages. Fig 6 shows that if a system is using the reactive maintenance policy, lookup costs are lowest when (for an $r$ value of 100), the $S_2$-type nodes send messages every 45 seconds on average, and do a successor stabilisation every 180 seconds on average. These results are not at all obvious and arise purely from the analysis.

**VI. SUMMARY**

In summary, we have demonstrated the usefulness of the master-equation approach for understanding churn in overlay networks. Our analysis can take into account most details of the algorithms used by these networks, to provide predictions for how the performance depends on the parameters. There are several directions in which we can extend the present analysis. One of the more important ones is to model congestion on the links. This could affect the performance of the two compared maintenance strategies differently. The periodic case may not be as affected as much as the reactive case, which could suffer from congestion collapse.

**Acknowledgments** We would like to thank Ali Ghodsi for several very useful discussions.
After some rewriting of this, it is easily seen that the cost for any key $i + 1$ can be written as the following recursion relation:

$$C_{i+1} = \rho C_i + (1 - \rho) + (1 - \rho) C_{i+1-\xi(i+1)}$$

(11)

Here we have used the definition of $a$ and $b$ from the internode-interval distribution and the notation $\xi(i+1)$ refers to the start of the finger most closely preceding $i + 1$. For instance, for $i + 1 = 4$, $\xi(i+1) = 2$ and for $i + 1 = 11$, $\xi(i+1) = 8$ etc.

We are interested in solving the recursion relation and computing $L = \sum_{i=1}^{K-1} C_i$. To do this, we decompose this sum into the following partial sums:

$$s_0 = C_1$$
$$s_1 = C_2$$
$$s_2 = C_3 + C_4$$
$$s_3 = C_5 + C_6 + C_7 + C_8$$
$$\vdots$$
$$s_M = C_{2M-1+1} + \ldots + C_{K-1}$$

(12)

Substituting the expressions for the $C$'s in the above, we find:

$$s_0 = 1$$
$$s_1 = \frac{\rho}{1 - \rho} [C_1 - C_2] + 1 + s_0$$
$$s_2 = \frac{\rho}{1 - \rho} [C_2 - C_4] + 2 + [s_0 + s_1]$$
$$\vdots$$
$$s_i = \frac{\rho}{1 - \rho} [C_{2i-1} - C_{2i}] + 2^{i-1} + \sum_{j=0}^{j-1} s_j$$

(13)
Thus, the equation for the average lookup length without churn is
\[
L = \frac{1}{2} M - \frac{\rho}{1 - \rho} \left[ C_2 - \frac{1}{K} \sum_{i=1}^{M-1} C_{2i} + \frac{1}{K} C_{K-1} \right] - \sum_{j=2}^{M-2} \frac{2 M - j - 1}{K} C_{2j-1}
\]
Eq. (17)

If we can take the limit \( K \to \infty \), we can throw away some of the terms.

\[
\lim_{K \to \infty} L = 1 + \frac{1}{2} M + \frac{\rho}{1 - \rho} \left[ C_2 - \frac{1}{K} \sum_{i=1}^{M-1} C_{2i} + \frac{1}{K} C_{K-1} \right]
\]
Eq. (18)

Since \( C_1 = 1 \), we can write

\[
L = 1 + \frac{1}{2} M + \frac{\rho}{2(1 - \rho)} \left[ \frac{C_2 - 1}{2} + \frac{C_4 - 1}{4} + \ldots \right]
\]
Eq. (19)

From the recursion relation for the \( C_i \)'s, it is easy to see that

\[
(C_1 - 1) = (1 - \rho) g_1^{(1)}(\rho) + (1 - \rho)^2 g_2^{(2)}(\rho) + \ldots
\]
Eq. (20)

where the \( g_i \)'s are functions only of \( \rho \).

Hence if \( (1 - \rho) \) is small \( (\frac{K}{N} \to 0) \), we need only compute the \( C_i \)'s to first order in \( (1 - \rho) \) to get the leading order effect and second order in \( (1 - \rho) \) to get the correction etc.

Hence in general the, the expression for \( L \) is

\[
L = 1 + \frac{1}{2} M + \frac{\rho}{2} \left[ e_1(\rho) + (1 - \rho) e_2(\rho) + (1 - \rho)^2 e_3(\rho) \ldots \right]
\]
Eq. (21)

Where \( e_1(\rho) = \sum_{i=1}^{M-3} g_2^{(1)}(\rho) \) etc.

We evaluate this expression numerically by solving recursion relation (11) and compare it with simulations done at zero churn. As can be seen the prediction of the equation is very accurate (Figure 7).

Let us now compute \( e_1(\rho) \) to see what the leading order effect is. We now need to solve recursion relation (11) only to order \( 1 - \rho \), which gives:

\[
C_2 - 1 = (1 - \rho)
\]

\[
C_4 - 1 = (1 - \rho) \left[ 1 + \rho + \rho^2 \right]
\]

\[
C_6 - 1 = (1 - \rho) \left[ 1 + \rho + \rho^2 + \ldots + \rho^6 \right]
\]

\[
\ldots
\]

\[
C_i - 1 = (1 - \rho) \left[ 1 + \rho + \rho^2 + \ldots + \rho^{i-2} \right]
\]
Eq. (22)

Therefore,

\[
L = 1 + \frac{1}{2} M + \frac{\rho}{2} \left[ 1 + \frac{\rho + \rho^2}{4} + \ldots \right]
\]
Eq. (23)

Consider the expression inside the brackets. We are computing this in the approximation \( \frac{N}{K} = \epsilon \to 0 \), i.e. \( \rho = 1 - \epsilon \), therefore \( \rho^x = (1 - \epsilon)^x \approx e^{-\epsilon x} \). If \( x > \frac{1}{\epsilon} \), then \( \rho^x \to 0 \), therefore if \( x > \frac{K}{N} \), then \( \rho^x \to 0 \). Hence, the terms inside the brackets become:

\[
\sum_{j=1}^{T} \frac{2^j - 1}{2^j} + (2^T - 1) \sum_{j=T+1}^{M-3} \frac{1}{2^j}
\]
Eq. (24)

Where \( T \equiv \ln_2 K - \ln_2 N \) and we have put \( \rho^x \approx 1 \) for \( x < \frac{K}{N} \) and \( \rho \to 0 \) for \( x > \frac{K}{N} \). This is clearly an overestimation and so we expect the result to over estimate the exact expression 21.

Expression 24 becomes:

\[
T - \left[ 1 - \left(\frac{1}{2}\right)^{M-3} \right] + \left[ 1 - \left(\frac{1}{2}\right)^{M-3-T} \right] \approx T
\]

Therefore:

\[
L = 1 + \frac{1}{2} \ln_2 K - \frac{1}{2} \left[ \ln_2 K - \ln_2 N \right]
\]
Eq. (25)

\[
\approx 1 + \frac{1}{2} \ln_2 N
\]

Which is the known result for the average lookup length of Chord.

Another important parameter in the performance of DHTs in general is the base. By increasing the base, the number of fingers per node increases which leads to a shorter lookup path.
length. The effect of varying the base has been studied in [3], [10]. So far, we have considered in this analysis base-2 Chord. We can likewise carry out this analysis for any base.

In general, we have base-\(b\) with \((b - 1)\log_b(K)\) fingers per node. Consider as an example \(b = 4\). Here we can define the partial sums again in the following manner:

\[
\begin{align*}
\Delta_0 &= s_0 = C_1 = 1 \\
\Delta_1 &= s_1 + s_2 + s_3 \\
\Delta_2 &= s_4 + s_5 + s_6 \\
&\ldots
\end{align*}
\]  

(26)

where

\[
\begin{align*}
s_1 &= C_2 = \rho C_1 + (1 - \rho) + (1 - \rho) C_1 \\
s_2 &= C_3 = \rho C_2 + (1 - \rho) + (1 - \rho) C_1 \\
s_3 &= C_4 = \rho C_3 + (1 - \rho) + (1 - \rho) C_1 \\
s_4 &= C_5 + C_6 + C_7 + C_8 \\
s_5 &= C_9 + C_{10} + C_{11} + C_{12} \\
s_6 &= C_{13} + C_{14} + C_{15} + C_{16} \\
&\ldots
\end{align*}
\]  

(27)

Therefore

\[
\begin{align*}
\Delta_0 &= C_1 \\
\Delta_1 &= \rho \left[ \Delta_1 + C_1 - C_4 \right] + 3(1 - \rho) + 3(1 - \rho) [\Delta_0] \\
\Delta_2 &= \rho \left[ \Delta_2 + C_4 - C_{16} \right] + 12(1 - \rho) + 3(1 - \rho) [\Delta_0 + \Delta_1] \\
&\ldots
\end{align*}
\]  

(28)

In general for a base \(b\), define \(B \equiv b - 1\) and \(b^M = K\). Then we have:

\[
\begin{align*}
\Delta_j &= \frac{\rho}{1 - \rho} [C_{b^j-1} - C_{b^j}] \\
&\quad + B(B + 1)^{j-1} + B \left[ \Delta_0 + \Delta_1 + \cdots + \Delta_{j-1} \right] \\
&\quad + B(B + 1)^{j-1} + B \left[ \Delta_0 + \Delta_1 + \cdots + \Delta_{j-1} \right]
\end{align*}
\]  

(29)

Following much the same procedure as before, we find

\[
L = \frac{1}{K} \sum_{j=0}^{M} \Delta_j \approx 1 + \frac{B}{B + 1} M - \frac{B}{B + 1} \frac{\rho}{1 - \rho} \left[ \frac{C_b - 1}{B + 1} + \frac{C_{b^2} - 1}{(B + 1)^2} + \ldots \right]
\]  

(30)

for \(K \to \infty\) as the analogue of (19). Again we can simplify and slightly overestimate the sum by assuming that \(\rho^x \approx 0\) for \(x > \frac{1}{b}\) and \(\rho^x \approx 1\) for \(x < \frac{1}{b}\). Then we get:

\[
L \approx 1 + \frac{b - 1}{b} \frac{\ln_2 N}{\ln_2 b}
\]  

(31)

This is the analogue of Eq. 25 for any base \(b\).