University of Würzburg
Institute of Computer Science
Research Report Series

Estimating Churn in Structured P2P Overlay Networks

Andreas Binzenhöfer[1] and Kenji Leibnitz[2]

Report No. 404 April 2007

[1] University of Würzburg, Institute of Computer Science
Chair of Distributed Systems, Würzburg, Germany
Email: binzenhoefer@informatik.uni-wuerzburg.de

[2] Osaka University, Graduate School of Information
Science and Technology, Suita, Osaka, Japan
Email: leibnitz@ist.osaka-u.ac.jp

Estimating Churn in Structured P2P Overlay Networks

Andreas Binzenhöfer
University of Würzburg, Institute of Computer Science
Chair of Distributed Systems, Würzburg, Germany
Email: binzenhoefer@informatik.uni-wuerzburg.de

Kenji Leibnitz
Osaka University, Graduate School of Information
Science and Technology, Suita, Osaka, Japan
Email: leibnitz@ist.osaka-u.ac.jp

***Abstract*— Structured peer-to-peer (P2P) networks have become an important alternative to the classic client-server architecture. Participating peers can join and leave the system at arbitrary times, a process which is known as churn. Many recent studies revealed that churn is one of the main problems faced by any Distributed Hash Table (DHT). As a countermeasure most DHT protocols have to invest a significant amount of overhead to maintain both the structure of the overlay and the redundancy of the stored data.**

In order to automatically adapt this maintenance overhead and other parameters of the DHT to the changes in the overlay, an estimation of the current churn rate is required. In this paper we discuss different possibilities of how to estimate the current churn rate in the system. In particular, we show how to obtain a robust estimate which is independent of the implementation details of the DHT. We also investigate the trade-offs between accuracy, overhead, and responsiveness to changes.

I. Introduction

With the recent development of new peer-to-peer (P2P) architectures, P2P has evolved from simple file-sharing networks to efficient content distribution platforms. Eliminating the potential dangers of overloaded servers in conventional client server systems at flash crowd arrivals, P2P poses as an efficient alternative for cooperatively storing and sharing information in the network. This is accomplished by each peer participating in a logical overlay structure and simultaneously acting as client and as server. The peer is responsible for maintaining its share of information and providing it to the other peers requesting this data.

Additionally, P2P networks have no static network topology and each participating peer may join or leave the overlay at any time. This process is referred to as *churn* [1]. However, this freedom of having a highly dynamic network structure comes at a cost. The higher the rate is at which peers join or leave the network, the more difficult it becomes for the network to maintain its consistency. A too high churn rate can cause loss of stored resources, routing failures, loss of the entire overlay structure, or inconsistent views of the peers on the overlay structure.

Thus, it is essential that the overlay network structure can be maintained even in the presence of a high churn rate. Especially, in structured P2P architectures, such as Chord [2], where all peers are arranged in a ring structure, the integrity of the neighborhood relationship among the peers must be kept at all times. Therefore, these networks require more maintenance traffic when the churn rate is high. However, usually P2P networks operate without a centralized control unit and each peer has only a limited view of the entire network, usually not being aware of the current churn rate in the network. Thus, a peer should be able to estimate the churn rate from the limited information that is available to it and then autonomously react to situations with high churn by increasing the maintenance traffic.

In this paper, we propose an entirely distributed algorithm for peers to estimate the churn rate by exchanging measurement observations among neighbors. The overlay network itself is used as a memory for the estimate while each online peer contributes to updated measurements of the estimator. The advantage of this method is that it operates passively, i.e., there are no additional entities required to monitor online and offline periods of the peers, and there is no further overhead necessary. While we mainly consider Chord-based DHT networks in this paper, our method is not restricted to any type of structured P2P network since it operates independently of the underlying DHT protocol. Wherever necessary, we will point out the corresponding differences to other types of structured P2P networks, e.g. Kademlia [3] or Pastry [4].

The paper is organized as follows. In Section II, we discuss some existing models for estimating the churn rate in P2P networks. This is followed by Section III where we give a detailed description of our proposed estimation scheme. Section IV will show that our proposed estimation scheme is capable of retrieving accurate estimates and we will study the impact of the parameters, e.g. the number of monitored neighbors or the stabilization interval, on the performance of our approach. Finally, we conclude the paper in Section V and elaborate on possible extensions.

II. Discussion of Different Churn Models

In P2P networks the term churn is used to describe the continuous process of new peers entering a P2P system and already participating peers leaving it. These fluctuations in the overlay can cause inconsistencies, lost messages, decreased user-perceived quality, or increased overhead. In this context, the impact of joining peers is usually less problematic, since it mainly results in temporary failures like routing inconsistencies or resources which might be temporarily located at a wrong position in the overlay. The process of peers leaving the

system, however, can result in irreparable damage like loss of the overlay structure or loss of data stored in the overlay. In general, node departures can be divided into *friendly leaves* and *node failures*. Friendly leaves enable a peer to notify its overlay neighbors to restructure the topology accordingly. Node failures, on the other hand, seriously damage the structure of the overlay by causing stale neighbor pointers or data loss. In this paper we concentrate on node failures.

In literature, there are two predominant ways to model churn. The first assumes churn per network by specifying a global join and leave rate [1]. This is also very similar to the half-life of a system as defined by Liben-Nowell et al. [5]. Usually the global join process is modeled by a Poisson process with rate λ in such a way that the time T_{join} between two join events is exponentially distributed. The problem with this model is that the number of nodes joining the system within a given time interval is independent of the current size of the system. However, while a join rate of 50 peers per second is quite significant for small networks, it might have no noticeable influence in very large networks.

Another way to model churn is to specify a distribution for the time a peer spends in the system (online time) or outside the system (offline time). This way the churn rate can be considered per node and thus generates a churn behavior, which is comparable in networks of different size. As in [6] we turn our main attention to scenarios where the join and failure rate are both described per node. To be able to model the offline time of a peer, we assume a global number of n peers, each of which can either be online or offline. Joins are then modeled by introducing a random variable T_{off} describing the duration of the offline period of a peer. Accordingly, leaves are modeled by a random variable T_{on} describing the online time of a peer. Usually, T_{on} and T_{off} are exponentially distributed with mean $E[T_{on}]$ and $E[T_{off}]$, respectively. However, this may not hold in realistic scenarios where distributions tend to become more skewed [7]. Therefore, in Section III we design our estimator in such a way that it is independent of the distribution of T_{on} and T_{off}.

The actual user behavior in a real system heavily depends on the kind of service which is offered. For example, Gummadi et al. [8] showed that P2P users behave essentially different from web users. For a typical filesharing application, they found a median session time of only 2.4 minutes and a 90th percentile of 28.25 minutes for sessions during which a peer was actively retrieving files. Their findings for large requests, however, showed that less than 10 percent are completed in an hour, 50 percent take more than a day, and nearly 20 percent of users are willing to wait a week for their downloads to complete. Additionally, Bhagwan et al. [9] argue that availability is not well-modeled by a single-parameter distribution, but instead is at least a combination of two time-varying distributions. This is supported by the observation that failure rates vary significantly with both daily and weekly patterns and that the failure rate in open systems is more than an order of magnitude higher than in a corporate environment [10].

Finally, to be able to compare the performance of different

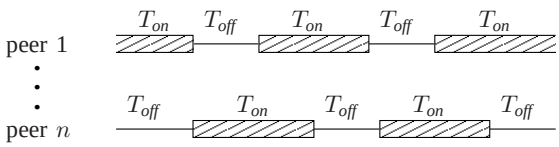

Fig. 1. T_{on} and T_{off} describe the online and offline times of the n peers.

selection strategies for overlay neighbors, Godfrey et al. [7] present a definition of churn which reflects the global number of changes within a time interval Δt:

$$C = \frac{1}{\Delta t} \cdot \sum_{events\ i} \frac{|U_{i-1} \ominus U_i|}{\max\{|U_{i-1}|, |U_i|\}}.$$

Thereby U_i is the set of online nodes which are in use after the ith change and $\ominus$ is the symmetric set difference. While the definition is very useful in simulations which possess a global view on the system, it cannot be used by an estimator which can only rely on local information.

III. Estimating the Churn Rate

In this section we describe our system model in greater detail and discuss the possibilities of how an individual peer can estimate the churn in the system. In general, a good estimator for the churn in the system must in some way capture the fluctuations in the overlay structure and then deduce an estimate for the churn rate from these observations. Thereby, we must take into account that an individual peer does not have any global knowledge about the state of the system but has to rely on a very limited view of the network. In structured P2P networks, each peer has periodic contact to a specific number of overlay neighbors, which are stored in a list. Those overlay neighbors are called *successors* in Chord, *k-bucket entries* in Kademlia, or *leafs* in Pastry. The basic principle of the estimator, which will be described in the following, is to monitor the changes in this neighbor list and use them to derive the current churn rate. The more observations a peer is able to collect, the more accurate its estimate is going to be. However, if it maintains too many entries in its observation history the estimate will respond very slowly to changes in the churn rate.

A. Obtaining Observations

As stated in Section II, the main focus of this paper is on churn per node. That is, we model the behavior of a peer using two random variables T_{on} and T_{off} which describe the duration of an online session and an offline session as shown in Figure 1.

It is reasonable to model the leave rate per node, since a peer can easily observe other online peers and their online session times. A join rate per node, however, depends on the fact that offline peers will rejoin the overlay network at a later point in time. While this is a very reasonable assumption for closed groups like company networks or distributed telephone directories (Skype), other applications like content distribution

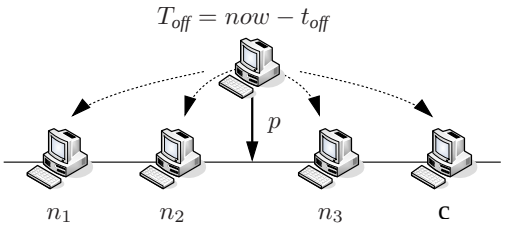

Fig. 2. Peer p rejoins the network and sends its offline duration to its c neighbors

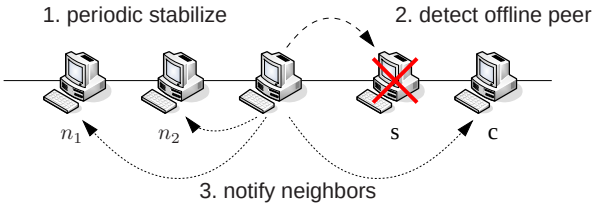

Fig. 3. Peer p only monitors its direct neighbor s but distributes its observations

(BitTorrent) might have no recurring customers. For the latter case, Ghinita et al. [11] present an estimator for the global join rate λ which is based on the average age of peers in the neighbor list. The main problem with such estimators is that they require an additional estimate of the current system size [12]. Furthermore, the accuracy of the estimates heavily depends on properly functioning neighbor update mechanisms.

In our model, we assume that each online peer p stores pointers to c well defined overlay neighbors (or contacts) which are specified by the individual DHT protocols. To deal with stale entries and to maintain the structure of the overlay, peer p periodically contacts a special subset of its neighbors every t_{stab} seconds and runs an appropriate *stabilization* algorithm. This corresponds, e.g., to *bucket refreshes* in Kademlia or the stabilization with the direct successor in Chord. At each of these stabilization instants the peer synchronizes its neighbor list with those of its contacts. The main idea of our estimator is to monitor the changes in the neighbor list and thereby collect different realizations of the random variables T_{on} and T_{off}. That is, a peer observes the online and offline session times of its overlay neighbors. Thereby, $obs(i)$ is the value of the ith observation made by the peer and $time(i)$ is the time at which the observation was made. The observation history is stored in a list which contains up to k_{max} entries. Furthermore, a peer stores the time stamps t^p_{on} and t^p_{off} which correspond to the time peer p itself joined or departed from the overlay, respectively. In the following we give a more detailed description of how a peer makes these observations and how it deduces estimates for both the join and the leave rate.

As discussed in Section II the join rate is the less important one of the two components of churn. The shorter a peer stays offline on average (i.e. the smaller $E[T_{off}]$) the higher is the join rate. To obtain realizations of T_{off}, a peer stores the time t_{off} when it last went offline. The next time it goes online it calculates the duration of its offline session as $now - t_{off}$ and sends this value to its c overlay neighbors. Figure 2 visualizes this concept. Note that the information can be piggybacked on other protocol messaged to avoid unnecessary overhead. For example, each time a peer joins the overlay network it has to contact its overlay neighbors to introduce itself to the network. In Chord, a joining peer contacts its successors and possibly its fingers, in Pastry its leaf set or neighborhood set, and in Kademlia it refreshes its closest bucket. These messages can be used to disseminate the observed offline time to the overlay neighbors.

Since failed nodes can no longer inform their overlay neighbors about their online duration, we are not in the position to directly obtain realizations of T_{on}. That is why we are looking for another passive way to collect realizations which is still independent of the applied DHT protocol. In a DHT system, a peer p periodically contacts at least one neighbor s to stabilize the overlay structure (cf. Step 1 in Figure 3). In Chord this would be the direct successor in a clockwise direction, in Kademlia the closest peer according to the XOR-metric. If, during one of its stabilization calls, p notices that s has become offline (cf. Step 2 in Figure 3), it calculates the duration of the online session of peer s as $now - t^s_{on}$, where t^s_{on} is the time when peer s went online. Peer p then distributes this observation to all its overlay neighbors as shown in Step 3 in Figure 3. If the DHT applies some kind of *peer down alert mechanism* [1], [10], the information could also be piggybacked on the corresponding notify messages.

An obvious problem of this approach is that peer p does not always naturally know t^s_{on}, the time when peer s went online. This is for example true if p went online after s or if s became the successor of p due to churn in the network. For this reason each peer s memorizes the time t^s_{on} when it went online and sends this information to its new predecessor whenever it stabilizes with a new peer. To cope with the problem of asynchronous clocks it sends its current online time $now - t^s_{on}$. This way the error is in the order of magnitude of a network transmission and thus negligible in comparison to the online time of a peer. The advantage of this method to collect realizations of T_{on} is that it only requires regular contact to one single neighbor.

When a peer joins the network, it first needs to obtain some observations before it can make a meaningful estimate of the churn rate. The problem is that the lifetime of the peer is in the same order of magnitude as the lifetime of its overlay neighbors. Thus, it is unlikely, that the peer is able to make enough observations during its lifetime. Therefore, we use the overlay network as a memory of already obtained observations to maintain them beyond the lifetime of the peer. If a new peer joins the overlay it downloads the current list of observations from its direct successor. This way the observations persist in the overlay and a new peer can already start with a useful estimate which reflects the current churn rate in the network.

Another way to maintain the persistence of the observations is to invest more overhead by periodically contacting a number of peers instead of just one. Mahajan et al. [13] present an

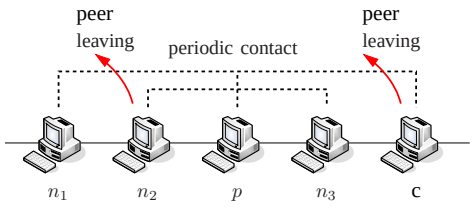

Fig. 4. Peer p periodically monitors the changes in its overlay neighborhood

algorithm which relies on the fact that a peer continuously observes c overlay neighbors as shown in Figure 4. Such a peer should on average observe one failure every $\Delta t = \frac{1}{c} \cdot E[T_{on}]$. Thus, if a peer observes k failures in Δt the mean online time of a peer can be estimated as:

$$\widehat{E}[T_{on}] = \frac{c \cdot \Delta t}{k} = \frac{c \cdot (time(k) - time(1))}{k}$$

where $time(i)$ is the time of the ith observed node failure. In addition to the periodic contact to c neighbors, the algorithm also has to struggle with the problem of obtaining enough observations during the lifetime of the peer. A possible solution to the problem is to piggyback the current estimate on protocol messages and to set the own estimate to the median of the estimates received from other nodes in the overlay [10].

B. Derivation of the Churn Rate

In this section we discuss the possibilities of a peer to deduce the current churn rate from the observations it has made. We will use the following notation: For a random variable X, we denote $x(t)$ as the probability density function, $X(t)$ as the cumulative density function, and $E[X]$ as the mean. Estimated values will be marked using a hat as in $\widehat{E}[X]$, which describes an estimate for the mean of X.

Once a peer has obtained a list of observations $obs(i), i = 1, \ldots, k$ of the random variables T_{on} and T_{off}, it needs a mechanism to derive an estimate of the current churn rate based on its a priori knowledge. If it cannot make any reasonable assumptions about the distribution of the random variable, it has to rely on robust estimates like the empirical mean and the empirical standard deviation.

$$\widehat{E}[T_{on}] = \frac{1}{k} \cdot \sum_{i=1}^{k} obs(i) \tag{1}$$

$$\widehat{\sigma}(T_{on}) = \sqrt{\frac{1}{k-1} \sum_{i=1}^{k} \left(obs(i) - \widehat{E}[T_{on}] \right)^2} \tag{2}$$

To evaluate the accuracy of the estimate we can construct the $100(1-\alpha)$ percent confidence interval for the estimated mean of T_{on} as

$$u(k,\alpha) = \widehat{E}[T_{on}] + t_{k-1,1-\frac{\alpha}{2}} \cdot \frac{\widehat{\sigma}(T_{on})}{\sqrt{k}} \tag{3}$$

$$l(k,\alpha) = \widehat{E}[T_{on}] - t_{k-1,1-\frac{\alpha}{2}} \cdot \frac{\widehat{\sigma}(T_{on})}{\sqrt{k}} \tag{4}$$

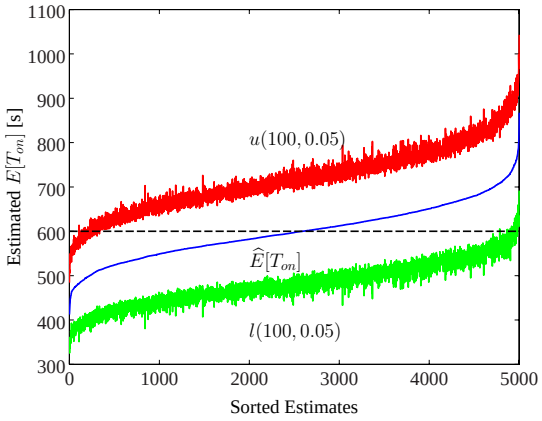

Fig. 5. Sorted estimates for $k = 100$ and $\alpha = 0.05$

where $t_{k-1,1-\frac{\alpha}{2}}$ is the $1 - \frac{\alpha}{2}$ critical point of the t distribution with $k-1$ degrees of freedom. Depending on the intended purpose of the estimator, it might be crucial that the estimator does not over- or underestimate the actual value too often. In such a case, the upper and the lower bound of the confidence interval can themselves be used as estimates. To clarify this concept, we simulated an overlay network with a mean online time of $E[T_{on}] = 600$s. Figure 5 plots the sorted estimates $\widehat{E}[T_{on}]$ of 5000 peers and the corresponding 95 percent confidence intervals based on $k = 100$ observations. While the regular estimates lie symmetrically around the actual value, the upper bound $u(k,\alpha)$ tends to overestimate and the lower bound $l(k,\alpha)$ tends to underestimate. Thereby the confidence level can be used to adjust the frequency at which the upper bound underestimates and the lower bound overestimates, respectively.

In general, the larger we set k, i.e. the more observations a peer maintains in its history, the more accurate the estimate is going to be. However, if k is chosen too large, it will take longer for the estimator to react to changes in the current churn rate. In this context, the limits of the confidence interval can be used to autonomously derive an optimal value of k. If the calculated confidence interval is larger than a predefined threshold, a peer can increase k accordingly.

While the mean gives a first idea about the churn in the system, the main purpose is to use the estimate to self-tune the parameters of the DHT or to calculate the probability of certain events. This usually requires knowledge of the entire distribution or at least of some important quantiles. For example, to calculate the probability that an overlay neighbor will no longer be reachable at the next stabilization instant, we need to know the probability that this contact will stay online for less than t_{stab} seconds. An unbiased point estimator for this probability is given by Equation 5.

$$\begin{aligned} \widehat{p} &= \widehat{P}\left(T_{on} < t_{stab}\right) \\ &= \frac{1}{k} \left| \left\{ T_{on}^i : T_{on}^i < t_{stab} \quad \text{for } i = 1, 2, ..., k \right\} \right| \end{aligned} \tag{5}$$

The $100(1-\alpha)$ confidence interval for $\widehat{p}$ can be calculated

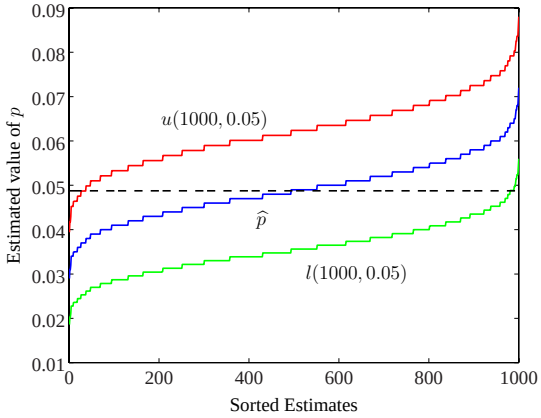

Fig. 6. Sorted estimates for $k = 1000$ and $\alpha = 95$

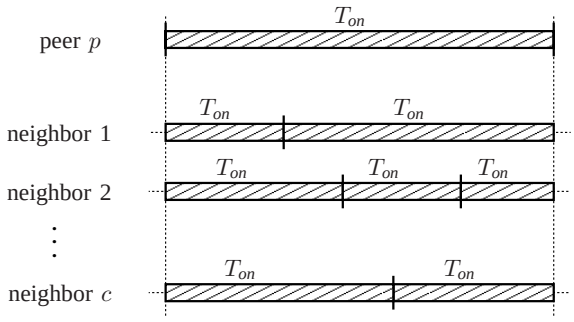

Fig. 7. Observations made by peer p during its lifetime

using the following bounds:

$$u(k,\alpha) = \widehat{p} + z_{1-\frac{\alpha}{2}} \cdot \sqrt{\frac{\widehat{p}(1-\widehat{p})}{k}} \tag{6}$$

$$l(k,\alpha) = \widehat{p} - z_{1-\frac{\alpha}{2}} \cdot \sqrt{\frac{\widehat{p}(1-\widehat{p})}{k}} \tag{7}$$

where $z_{1-\frac{\alpha}{2}}$ is the $1-\frac{\alpha}{2}$ critical point for a standard normal random variable. In case over- or underestimating has serious consequences for the applied application, the limits of the confidence interval can again be used as estimates themselves.

We simulated an overlay network with $t_{stab} = 30$s where the online time of a peer was exponentially distributed with mean $E[T_{on}] = 600$s. Under these conditions, the probability p that a specific peer goes offline before the next stabilization instant is 4.88 percent. Figure 6 shows the sorted estimates of p and the corresponding upper and lower bounds from 1000 peers. As before, the upper bound $u(k,\alpha)$ tends to overestimate and the lower bound $l(k,\alpha)$ tends to underestimate. Note, that due to the denominator in Equation 5 the estimate is discretized into steps of $\frac{1}{k}$.

In some cases an application requires knowledge of the entire distribution function of the online time of the peers. If the type of distribution is known a priori, the peer can use the corresponding *Maximum Likelihood Estimator* (MLE) to estimate the corresponding parameters of the distribution. In the most often assumed case of an exponential distribution, the MLE is known to be the sample mean. For other typical distributions like the log-normal distribution Log-N(μ, σ^2) the MLE becomes more complicated, but can usually still be calculated using the information collected by the peer.

$$\widehat{\mu} = \frac{1}{k} \cdot \sum_{i=1}^{k} \ln\left(obs(i)\right) \tag{8}$$

$$\widehat{\sigma} = \left[\frac{\sum_{i=1}^{k} \left(\ln\left(obs(i)\right) - \mu\right)^2}{k}\right]^{\frac{1}{2}} \tag{9}$$

However, there is always the danger of assuming an incorrect distribution which would lead to correspondingly distorted results. A possibility to reduce this risk is to perform a hypothesis test [14] to verify that the type of distribution is actually the assumed one and only use an MLE if the test delivers a positive result. In general, however, the actual type of distribution is not known or a superposition of multiple distributions. In this case, a peer has to rely on an estimate of the quantiles [15] of the online distribution. Let $T_{on}^1, T_{on}^2, ..., T_{on}^k$ be the k observations in the history of a peer and let $T_{on}^{(1)}, T_{on}^{(2)}, ..., T_{on}^{(k)}$ be the ordered statistic, in such a way that $T_{on}^{(1)} < T_{on}^{(2)} < ... < T_{on}^{(k)}$. These sorted values of T_{on} can than be taken as the $\frac{0.5}{k}, \frac{1.5}{k}, ..., \frac{k-0.5}{k}$ quantiles of the distribution of the online time. Quantiles for probabilities between $\frac{0.5}{k}$ and $\frac{k-0.5}{k}$ can be computed using linear interpolation, while the minimum or maximum values of T_{on} are assigned to quantiles for probabilities outside that range.

The accuracy of the estimates heavily depends on k. The more observations a peer maintains in its history, the more accurate the estimate is going to be. However, if the overlay network is not used as a memory for already made observations, a joining peer has to rely on its own observations. Therefore, it can either observe one specific peer and send the result to its c overlay neighbors or directly observe c peers itself. Figure 7 shows the online period of a peer p and the c overlay neighbors it observes during its lifetime. In the figure we assume a perfect stabilization algorithm. That is, an overlay neighbor which went offline is immediately replaced by another overlay peer.

To analyze the expected size of the history of a peer, we regard the random variable X which describes the number of observations a peer makes during its lifetime. This number corresponds to the number of leave events in Figure 7. It can be computed as

$$P\left(X = i\right) = \int_0^\infty t_{on}(t) \cdot P\left(X = i | T_{on} = t\right)\, dt \tag{10}$$

where $t_{on}(t)$ is the probability density function of T_{on}. In the case of exponentially distributed online times, this can be written as

$$P\left(X = i\right) = \int_0^\infty \lambda e^{-\lambda t} \cdot \frac{(c\lambda t)^i}{i!} \cdot e^{-c\lambda t}\, dt \tag{11}$$

since the number of departures in a fixed interval of length t is Poisson distributed with $c \cdot \lambda$. The equation can be simplified

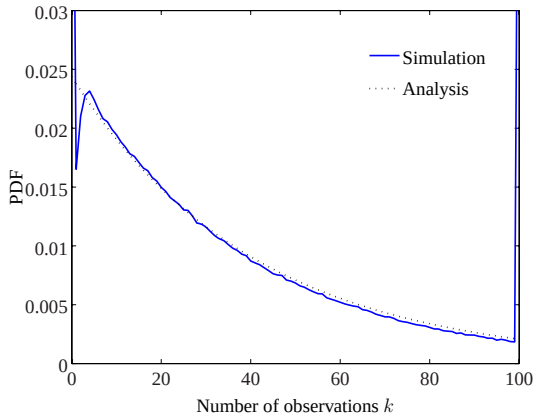

Fig. 8. Expected number of observations for $c = 40$

to

$$P(X = i) = \frac{c^i \lambda^{i+1}}{i!} \int_0^\infty t^i \cdot e^{-(c+1)\lambda t}\, dt = \frac{c^i}{(c+1)^{i+1}}. \quad (12)$$

To compare this theoretical approximation to practical values, we simulated an overlay network with $T_{on} = 300$s, $t_{stab} = 30$s, and $c = 40$. The maximum size of the history was set to $k_{max} = 100$. Figure 8 shows the probability density function of X for both the analysis and the simulation. It can be seen that the analysis matches the simulation very well except for the two peaks at the left and the right of the figure. The peak at 100 clearly results from the maximum size of the history. That is, all probabilities for $P(X > 100)$ are added to $P(X = 100)$. The peak at 0 arises from the fact that while the analysis immediately takes offline peers into account, the first stabilization instant in the simulation occurs 30 seconds after the peer joined the network. Thus, all peers which stay online for less than 30 seconds, can never make an observation.

In either case the results show that a peer does not make enough observations during its lifetime in order to derive a meaningful estimate. Also note that the figure already represents the best case since it illustrates the size of the history at the end of the online session of the peer. In order to achieve a higher accuracy, a good estimator should therefore utilize the overlay network as a memory for already made observations.

While so far we studied the accuracy of an estimator, it is also interesting to analyze how fast an estimator reacts to changes in the global churn rate. The more observations a peer makes per time unit, the faster it can react to such changes. This can be measured by looking at T_{obs}^{leave}, the time between two observed leave events, or T_{obs}^{join}, the time between two observed join events. In general, the collection of observations shows a self-organizing behavior. The more churn there is in the system, the more observations will be collected per time unit. If a peer shares its observations with c overlay neighbors, the next observation is made as soon as one of these $c + 1$ peers goes offline. Thus, the distribution of T_{obs}^{leave} can be calculated as the minimum of $c+1$ forward recurrence times of T_{on}. Due to the memoryless property, the forward recurrence time of an exponentially distributed online time T_{on} is also exponentially distributed with the same parameters. In this case the distribution of T_{obs}^{leave} can be calculated as shown below.

$$P\left(T_{obs}^{leave} < t\right) = 1 - P\left(T_{on} \geq t\right)^{c+1} = 1 - e^{-(c+1)\lambda t} \quad (13)$$

If the distribution is not known, we can still easily compute the mean of T_{obs}^{join} and T_{obs}^{leave}. Each node which joins the network calculates its own offline time and additionally sends this observation to its c contacts.

$$E\left[T_{obs}^{join}\right] = \frac{E[T_{off}]}{c+1} \quad (14)$$

The calculation is slightly more complicated for T_{obs}^{leave} since the time when a peer actually observes that another peer is offline differs from the actual time the node left the overlay. Assuming that overlay neighbors are updated every t_{stab} seconds, the average error is

$$\epsilon_{on} = \frac{t_{stab}}{2}. \quad (15)$$

Thus, the mean of T_{obs}^{leave} can be calculated as

$$E\left[T_{obs}^{leave}\right] = \frac{E[T_{on}] + \epsilon_{on}}{c+1}. \quad (16)$$

Obviously, the more overlay neighbors a peer maintains, the more observations it receives per time unit. However, more neighbors also correspond to more overhead in terms of bandwidth. This trade-off will be discussed in greater detail in Section IV. Also note that $E\left[T_{obs}^{join}\right]$ and $E\left[T_{obs}^{leave}\right]$ can be measured by an online peer and could therefore be used as estimates for $E[T_{off}]$ and $E[T_{on}]$, respectively.

The above considerations can be used to approximate the expected time it takes the estimator to respond to a global change of the churn rate. When the mean online time of the peers changes from $E_{old}[T_{on}]$ to $E_{new}[T_{on}]$, we approximate the expected response time $E[R]$ by the time needed to collect k_{max} new observations.

$$E[R] = E_{old}[T_{on}] + \frac{k_{max}}{c+1} \cdot \left(E_{new}[T_{on}] + \epsilon_{on}\right) \quad (17)$$

Figure 9 compares the analytical response time to that obtained from a simulation run. In the simulation we set $k_{max} = 100$, $c = 10$, $t_{stab} = 30$s and changed $E[T_{on}]$ from 10min to 5min to 15min and back to 10min after 8.33h, 16.66h, and 25h of simulation time, respectively. The simulated curve shows the mean of the estimated $E[T_{on}]$ values of all peers, which were online at the corresponding time. The error bars represent the interquartile range, i.e. the difference between the third and first quartiles, as a measure of statistical dispersion. It can be seen that the estimator is able to capture the changes in the churn rate and that the time it takes to adjust to the new value complies with the analysis. Note that, due to

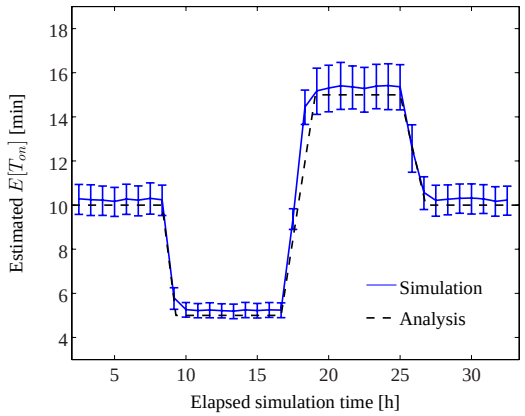

Fig. 9. Response time for $c = 10$ and $k_{max} = 100$

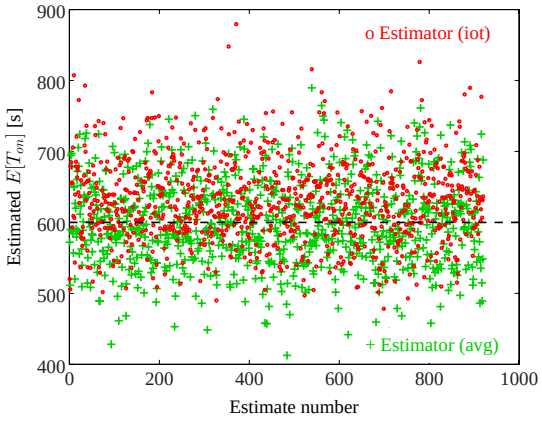

Fig. 10. Snapshot of mean online time obtained from *avg* and *iot* estimation

the stabilization period of 30 seconds, the estimated values lie $\epsilon_{on} = 15$s above the actual value.

The response time can be improved by maintaining more overlay neighbors or by giving less weight to older values in the history. This can, e.g., be achieved by using an exponential weighted moving average [16], which applies weighting factors which decrease exponentially.

$$\widehat{E}_k[T_{on}] = \alpha \cdot obs(i) + (1 - \alpha) \cdot \widehat{E}_{k-1}[T_{on}]$$

Thereby, the smoothing factor α determines the weight given to the latest observation.

IV. Numerical Results

In this section we will evaluate the proposed estimator using simulation results. The section is divided into three parts. First, we will provide a proof of concept (Section IV-A) and show the accuracy of our proposed method (IV-B). Beside accuracy one of the desired features of our proposal is a high responsiveness towards changes in the current churn rate. This will be discussed in Section IV-C. Finally, we will elaborate on some issues dealing with the practical implementation of the estimator in Section IV-D.

In the following, unless stated otherwise, we will always consider that the online and offline times of the users are exponentially distributed with mean $E[T_{on}]$ and $E[T_{off}]$, respectively. The default stabilization interval is $t_{stab} = 30$s and the size of neighbor list is $c = 20$. We will further assume that there are 40000 initial peers, resulting in an average of 20000 online peers at a time. Although our estimator yields results for both online and offline time, we will concentrate on estimating the online time T_{on}, since this is usually a more important parameter for the system performance. The estimation of T_{off} can be calculated in an analogous way.

A. Proof of Concept

The main purpose of this section is to show that the theoretic concept of the proposed estimator as described in Section III does work equally well in practice. We focus on Chord since it is the currently most studied DHT network architecture. In order to support the evaluations, we will provide additional analytical calculations verified by simulations. The simulations are not conducted modeling all the features of the Chord network in detail, but are simplified by only taking into account such properties which are important to our estimator. That is, we mainly disregard all mechanisms dealing with document management or replication. To model the stabilization algorithm, a peer synchronizes its neighbor list every $t_{stab} = 30$s with its direct successor. When a peer notices that another peer is offline, it notifies the peers in its neighbor list, piggybacking the observed online time in these messages. We furthermore consider a symmetric neighbor list, i.e. the number of peers in the successor list is the same as that of the predecessor list. This improves the stability of the Chord overlay and provides a better comparability of the result to symmetric overlays like Kademlia.

Figure 10 plots the estimated values for $E[T_{on}]$ obtained by two different estimation methods. The green crosses, denoted by *avg*, show the mean of the observed values as given by Equation 1. The red circles, denoted by *iot*, are based on the mean time between two observations according to Equation 16. The figure was created by picking 900 random peers leaving the overlay and calculating their estimates based on both estimation methods. We can see that both methods perform quite well, as the estimated values clump around the actual average T_{on} of 600s. In general the *avg* method always provides a robust estimate. The *iot* method, however, heavily depends on the implementation specifics of the stabilization routine. In our simulation experiments we do assume an idealized stabilization routine, neglecting packet loss and other (network) errors. In a real implementation the neighbor list can still contain wrong or offline entries after stabilization. In such a case the updates may be sent to less than c contacts. If a peer, however, assumes that c other peers contribute with their estimated values, it will receive too few estimates and heavily underestimate the actual churn rate using the *iot* method. Thus, while in theory it can be considered as a good method, it is in fact not very suitable in practice.

In practice too high or too low estimates might have critical

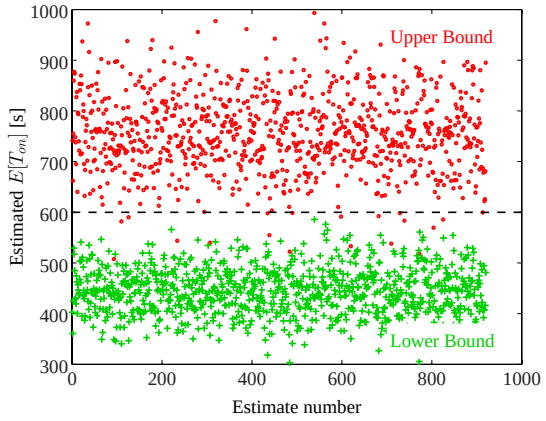

Fig. 11. Estimates of upper and lower 99% confidence levels

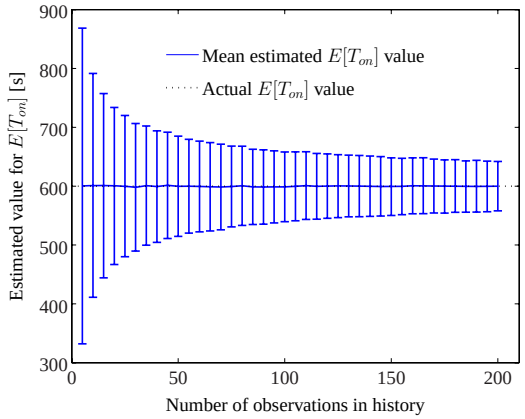

Fig. 12. Influence of the history size k on the estimation accuracy

consequences in terms of performance or even functionality. In such a case it should be avoided that the estimator underestimates or overestimates the actual churn rate. This can be achieved by using the upper or lower bound of a specified confidence level instead of the estimated value itself. Figure 11 shows the upper and lower bounds of the 99% confidence interval for the mean according to Equation 3 and Equation 4, respectively. As expected, the upper bound overestimates the actual value, while the lower bound underestimates it. The frequency at which the upper bound underestimates or the lower bound overestimates the actual value can be influenced by the confidence level. The higher the confidence level is chosen, the smaller is the probability for this to happen at the cost of more innacurate values.

B. Accuracy of the Estimator

In this section we evaluate how accurate the estimated results are by investigating the influence of different parameters. The key parameter we focus on is the size k of the observation history, i.e., the number of samples that are used to obtain the estimate.

In order to show the influence of k on the accuracy of the estimate, we perform several simulation runs in which we vary the size k of the history and for each k examine

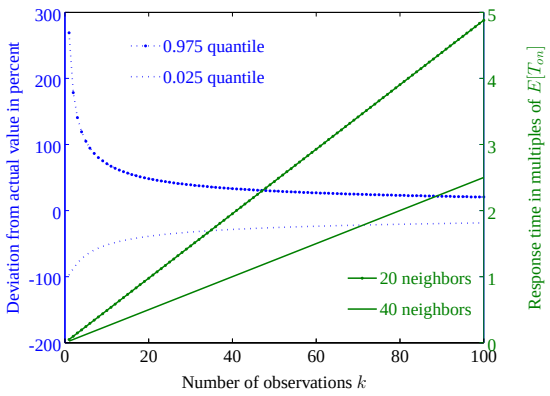

Fig. 13. Trade-off between accuracy and responsiveness

the estimated $E[T_{on}]$ values from 10000 peers. The mean of these estimated values is shown in Figure 12. The error bars in the figure represent the sampled standard deviation obtained from the 10000 estimates. First of all, we can recognize that the method is robust for estimating the mean, as the mean estimated $E[T_{on}]$ value corresponds to the actual mean value of 600s. Furthermore, increasing the history size greatly improves the accuracy of the estimate in terms of a smaller standard deviation. The accuracy improves nearly exponentially with k. However, increasing the history size to a too large value above $k = 100$ does not significantly improve the accuracy while it will lead to a slower responsiveness of the estimator as shown in the next paragraph. Therefore, in the following, we use a history size of $k = 100$ unless stated otherwise.

In the next step we take a closer look at the trade-off between accuracy and responsiveness in dependence of the size of the history. To express accuracy, we regard the deviation from the actual value in percent. In particular, we consider how much the 97.5% and 2.5% quantiles of the estimated values based on k observations differ from the actual value in percent. This is plotted as the dotted blue curves in Figure 13 using the left y-axis. As in the previous figure, it can be recognized that an increase in the history size results in more accurate estimates. The quantiles confirm the exponential dependence on k.

An increased accuracy, however, comes at the drawback of reducing the responsiveness of the estimator. Responsiveness is defined as the time it takes to collect k fresh results when there is a change in the global churn rate. It is expressed in multiples of $E[T_{on}]$ and approximated by Equation 17. The responsiveness increases linearly with k as can be seen in the green solid curves of Figure 13 using the y-axis on the right. The slope of the curve is determined by the number of overlay neighbors. The more neighbors there are, the more results are obtained per time unit and the faster the estimator reacts to the change in the churn rate. The study shows that depending on the application requirements, a trade-off can be made between a higher accuracy and a faster responsiveness by changing the number of considered observations.

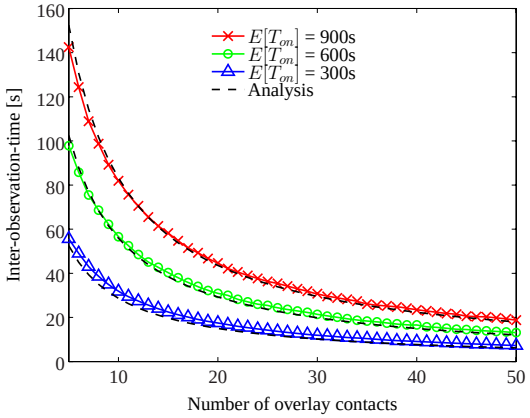

Fig. 14. Responsiveness to different churn rates

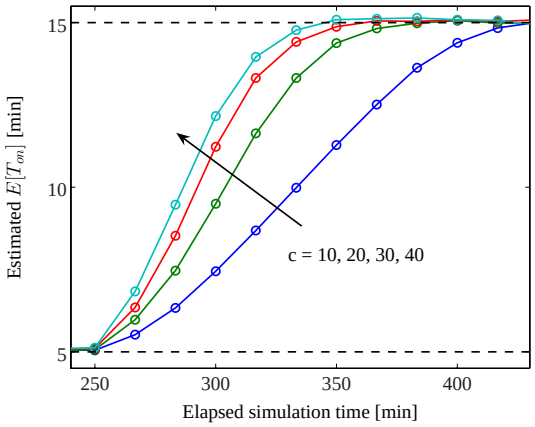

Fig. 15. Reaction to change in global churn rate

C. Responsiveness of the Estimator

In the previous section, we have already briefly discussed the issue of responsiveness. It is a measure for the time it takes our estimator to react to changes of the global churn rate of the network. It mainly depends on the number of overlay contacts which share their observations, but is also influenced by the absolute churn rate itself. The higher the churn rate is, the more results can be collected within the same time period.

In order to provide a more comprehensive study of the responsiveness of the estimator and to validate our analytical approximation in Equation 17, we perform simulation runs with different churn rates and measure the time between two successive observations. Obviously, the smaller this inter-observation time is, the faster our method will react to changes of the churn rate. This is shown in Figure 14. For different churn rates of $E[T_{on}] = 300$s, 600s, and 900s, the inter-observation time is depicted as a function of the number of overlay contacts. The dashed lines are the results obtained by the approximation, cf. Equation 17. It can be seen that the inter-observation time decreases exponentially and that the analytical curves match well with those obtained by simulations. However, we can also recognize that a greater number than 20 neighbors is not justified due to the small improvement in responsiveness and the higher overhead in maintaining those neighbors. Smaller values of $E[T_{on}]$, i.e. higher churn rates, result in smaller values of the inter-observation time. However, the number of overlay contacts has an even greater influence on the inter-observation time. Note, that the responsiveness also depends on the quality of the stabilization algorithm. If a simple algorithm is used, the neighbor lists might be inaccurate, which in turn results in a loss of updates and thus a higher inter-observation time.

To show how the inter-observation time translates into the actual response time and how the estimator behaves during these reaction phases, we simulated a network where the mean online time of all peers was globally changed from the initial value of 300s to 900s after a simulation time of 250 minutes. In Figure 15 each data point shows the average of the estimated $E[T_{on}]$ values of all online peers at the same time instant. The figure visualizes that the estimates react differently to the change of the global churn rate. Again the more neighbors there are, the faster the estimator approaches the new churn rate. While the increase is nearly linear for $c = 10$, there is not much difference between the curves for $c = 20$, 30, and 40 neighbors. Thus, using a too large number of overlay contacts is not justified due to the additional overhead. Using 20 overlay neighbors, as e.g. suggested in Kademlia, is therefore a reasonable choice.

D. Practicability and Implementation Aspects

So far, we studied the performance of the approach based on theoretical issues such as accuracy and responsiveness. However, in real implementations, other aspects will be even more important. In practice, the estimate of the churn rate will be used to self-adaptively tune maintenance algorithms of the P2P network, e.g. the stabilization of the overlay structure and the control of the replication of stored documents. Therefore, it would be desirable that all peers obtain equal estimate values in order to derive similar input parameters to these algorithms. Since our proposed estimation method is entirely distributed and each peer calculates its own estimate from local measurements, different peers also tend to obtain different estimation results. However, most maintenance algorithms are performed between direct overlay neighbors of the DHT.

1) Correlation Between Estimates of Overlay Neighbors: Since these direct overlay neighbors also exchange their measured observations, their churn estimates derived from this data are expected to be highly correlated. To quantify the degree of this correlation, we took a global snapshot during the simulation and had a closer look at the estimates of 5000 consecutive peers on the Chord ring. We then investigated the correlation between these peers by applying methods from time series analysis. Figure 16 depicts the autocorrelation over the number of neighbors. The x-axis represents the different lags x, which in our case corresponds to the x-th overlay neighbor. If x is positive, this corresponds to the x-th successor on the ring, whereas a negative value represents the $|x|$-th predecessor. The figure shows that the estimates of a peer are indeed highly correlated among neighboring peers. The

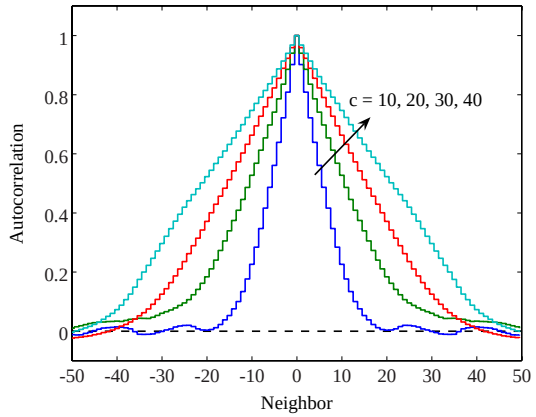

Fig. 16. Autocorrelation of estimates over neighboring peers

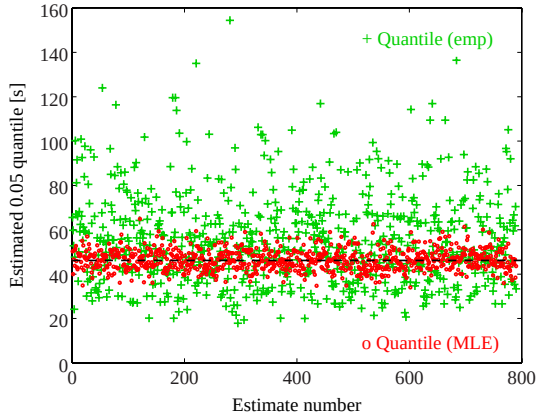

Fig. 17. Comparison of 0.05 quantile from MLE and empirical estimate

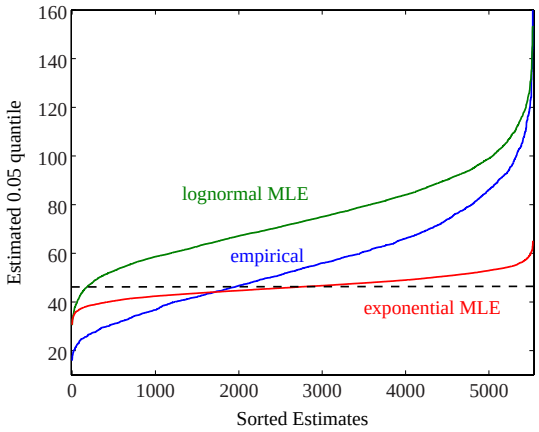

Fig. 18. Estimation based on empirical quantile and MLEs for exponential and lognormal dsitributions

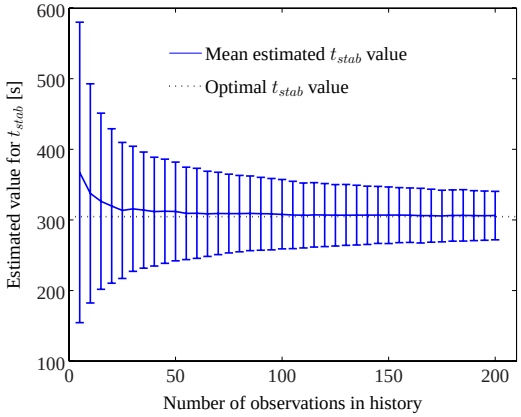

Fig. 19. Influence of history size on choice of stabilization interval

curves for the different numbers c of overlay neighbors among which the measurement values are exchanged show that the correlation extends to at least c neighbors in both directions of the ring. Note that due to our symmetric neighbor lists a value of $c = 10$ overlay neighbors means that the peer maintains 5 neighbors in both directions of the ring.

2) Influence of Assuming a Wrong Distribution: In practice, maintenance algorithms usually require quantiles instead of just average values of the churn rate. The following study considers simulation results of a network using a stabilization interval of 10s and an exponentially distributed T_{on} with mean $E[T_{on}] = 900$s. We now compare the estimation of the 0.05 quantile of the online time of a peer obtained by two different methods. In the first method we calculate the maximum likelihood estimator (MLE) of the exponential distribution and then take the 0.05 quantile. In the second method we simply use the empirical 0.05 quantile based on the 100 observations in the history of the peer. The exact value lies at 46s. In Figure 17 we can see that the MLE clearly outperforms the empirical estimate as the values lie much closer around the exact value. Obviously, this good result is due to the fact that the actual online time is really exponentially distributed and the MLE method thus uses a perfect assumption.

In real networks, however, this assumption is rarely valid. Therefore, we are interested in the results in case the underlying distribution is not known or a wrong assumption about the distribution is made. This is examined in another simulation run with the same parameters as before. This time the quantile is calculated using three different approaches: the empirical quantile and the quantiles based on the MLEs for the exponential and lognormal distributions. Figure 18 shows the estimates of the quantiles with the three methods sorted by size. Again the exponential MLE achieves the best results since the underlying assumption of an exponential distribution is correct. The lognormal MLE tries to estimate the parameters of a lognormal distribution while the actual distribution is exponential. It can be seen that this false assumption leads to a significant overestimation of the actual value. Therefore, when the actual distribution is unknown, which is usually the case in a real network, any assumption must be made with care. The validity of the underlying assumption can be tested by performing a hypothesis test where the appropriate MLE will only be used when the test is positive. In case the test is negative the empirical estimate should be preferred.

3) Self-Tuning the Stabilization Interval: The main purpose of the proposed estimator is expected to be the self-tuning

of the stabilization of the overlay structure. In practice, the stabilization interval, i.e. the frequency at which overlay neighbors are contacted to update the neighbor lists, is a fixed value. Thus, when there is no churn in the network this results in unnecessary overhead. However, when there is a high churn rate, the stabilization overhead may not be sufficient to maintain the stability of the overlay. For self-adaptive selection of the t_{stab} setting, a peer should therefore estimate the current churn rate and use this estimate to derive the probability that all neighbors will be offline before the next stabilization call. Such a scenario would result in the entire overlay structure becoming instable. For example, given a mean online time of $E[T_{on}] = 600$s, a peer needs to stabilize at least every 300s in order to maintain the overlay stability with a probability of 99.99%. In Figure 19, the stabilization interval t_{stab}, as derived based on an estimated churn rate, is shown over the size of the observation history. As before in Figure 12, the figure includes the mean and standard deviation. It can be seen that the standard deviation decreases exponentially and that a history size of 100 again results in a good value for practical purposes.

V. Conclusion

Structured P2P networks apply different maintenance mechanisms to guarantee the stability of the overlay network and the redundancy of stored documents. Ideally, the parameters of these mechanisms should be adapted to the current churn rate. The more churn there is in the system, the more overhead is needed to keep the system stable. As a first step toward a self-organizing overlay network, we introduced a method which enables a peer to estimate the current churn rate in the system. The estimate can then be used to autonomically adapt the overhead.

The estimator is based on the changes a peer observes in its list of overlay neighbors. The more observations a peer makes, the better is the quality of its estimate. Therefore, a peer shares observed events with its direct overlay neighbors by piggybacking the corresponding information in regular protocol messages. Both analytical and simulation results show that the estimator is able to capture the current churn rate. The accuracy, the required overhead, and the responsiveness to changes can be adjusted by the number of observations considered in the estimation process and by the number of overlay neighbors which share the results. We investigated the corresponding trade-offs and deduced values which are suitable for practical purposes. For applications, which are sensitive to an overestimation or underestimation of the actual value, we showed how to use the upper and lower bounds of a confidence interval as estimates themselves.

In future work, we intend to use the estimator to enable a peer to self-adapt the number of overlay neighbors and the number of replicas to the current churn rate. This way, the functionality of the overlay network will still be guaranteed in times of high churn while the maintenance overhead will be reduced in times of no churn.

Acknowledgments

The authors would like to thank Dirk Staehle and Simon Oechsner for their many ideas, the much appreciated input, and the insightful discussions during the course of this work.

References

[1] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT," in *2004 USENIX Annual Technical Conference*, (Boston, MA), June 2004.

[2] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *ACM SIGCOMM 2001*, (San Diego, CA), August 2001.

[3] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *IPTPS 2002*, (MIT Faculty Club, Cambridge, MA, USA), March 2002.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, (Heidelberg, Germany), pp. 329–350, November 2001.

[5] D. Liben-Nowell, H. Balakrishnan, and D. Karger, "Analysis of the Evolution of PeertoPeer Systems," in *ACM PODC*, (Monterey, CA, USA), July 2002.

[6] S. Krishnamurthy, S. El-Ansary, E. Aurell, and S. Haridi, "A statistical theory of chord under churn," in *The 4th Annual International Workshop on Peer-To-Peer Systems (IPTPS 05)*, (Ithaca, NY, USA), February 2005.

[7] P. B. Godfrey, S. Shenker, and I. Stoica, "Minimizing churn in distributed systems," in *Proc. of ACM SIGCOMM*, (Pisa, Italy), September 2006.

[8] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, (New York, NY, USA), pp. 314–329, ACM Press, 2003.

[9] R. Bhagwan, S. Savage, and G. Voelker, "Understanding availability," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[10] M. Castro, M. Costa, and A. Rowstron, "Performance and dependability of structured peer-to-peer overlays," in *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, (Washington, DC, USA), p. 9, IEEE Computer Society, 2004.

[11] G. Ghinita and Y. Teo, "An adaptive stabilization framework for distributed hash tables," in *Proceedings of 21st International Parallel & Distributed Processing Symposium, IEEE Computer Society Press*, (Rhodes Island, Greece), April 2006.

[12] A. Binzenhöfer, D. Staehle, and R. Henjes, "On the Fly Estimation of the Peer Population in a Chord-based P2P System," in *19th International Teletraffic Congress (ITC19)*, (Beijing, China), September 2005.

[13] R. Mahajan, M. Castro, and A. Rowstron, "Controlling the cost of reliability in peer-to-peer overlays," in *IPTPS 2003*, (Berkeley, CA, USA), February 2003.

[14] M. A. Stephens, "Edf statistics for goodness of fit and some comparisons," in *Journal of the American Statistical Association*, vol. 69, pp. 730–739, 1974.

[15] E. J. Chen and W. D. Kelton, "Quantile and histogram estimation," in *WSC '01: Proceedings of the 33nd conference on Winter simulation*, (Washington, DC, USA), pp. 451–459, IEEE Computer Society, 2001.

[16] J. Kurose and K. Ross, *Computer Networking A Top-Down Approach Featuring the Internet*. Addison Wesley Longman Inc., 2005.