Estimating Churn in Structured P2P Networks

Andreas Binzenhöfer[1] and Kenji Leibnitz[2]

[1] University of Würzburg, Institute of Computer Science
Chair of Distributed Systems, Würzburg, Germany
binzenhoefer@informatik.uni-wuerzburg.de
[2] Graduate School of Information Science and Technology
Osaka University, 1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
leibnitz@ist.osaka-u.ac.jp

Abstract. In structured peer-to-peer (P2P) networks participating peers can join or 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). 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.

1 Introduction

With the recent development of new peer-to-peer (P2P) architectures, P2P has evolved from simple file-sharing networks to efficient alternatives to the classic client-server architecture. This is accomplished by each peer participating in a logical overlay structure, 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 churn rate is, the more difficult it becomes for the network to maintain its consistency [2]. Too high churn can cause routing failures, loss of stored resources or the entire overlay structure, or inconsistent views of the peers on the overlay.

Thus, it is essential that the overlay network structure is maintained even in the presence of high churn. Especially in structured P2P architectures, such as Chord [3], where all peers are arranged in a ring structure, the integrity of the neighborhood relationship among the peers must be kept at all times. As a consequence, these networks require more maintenance traffic when the churn rate is high. However, 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 and autonomously react to high churn situations by increasing the maintenance traffic.

In this paper, we propose a fully 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 no further overhead is necessary. While we mainly consider Chord-based DHT networks, our method is not restricted to any type of structured P2P overlay 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 [4] or Pastry [5].

The paper is organized as follows. In Section 2, we discuss some existing models for estimating the churn rate in P2P networks. This is followed by Section 3 where we give a detailed description of our proposed estimation scheme. Section 4 will show that our algorithm 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 5 and elaborate on possible extensions.

2 Discussion of Different Churn Models

The impact of joining peers is usually the less problematic aspect of churn, 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 therefore concentrate on node failures.

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 in [6]. Usually the global join process is modeled by a Poisson process with rate λ. One of the main problems of 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 [7] 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 [8]. Therefore, in Section 3 we design our estimator 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. [9] showed that P2P users behave essentially different from web users. Additionally, Bhagwan et al. [10] 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 [11]. Finally, to be able to compare the performance of different selection strategies for overlay neighbors, Godfrey et al. [8] present a definition of churn which reflects the global number of changes within a time interval Δt. While the definition is very useful in simulations which permit a global view on the system, it cannot be used by an estimator which can only rely on local information.

3 Estimating the Churn Rate

In general, an 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. In structured P2P networks, each peer has periodic contact to a specific number of overlay neighbors. Those overlay neighbors are called *successors* in Chord, *k-bucket entries* in Kademlia, or *leafs* in Pastry. The basic principle of the estimator described here is to monitor the changes in this neighbor list and use them to derive the current churn rate.

3.1 Obtaining Observations

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. This model assumes 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 corporate networks or distributed telephone directories (Skype), other applications like content distribution (BitTorrent) might have no recurring customers. For the latter case, an estimator for the global join rate λ is presented in [12] based on the average age of peers in the neighbor list. The main problem is that such estimators require an additional estimate of the current system size [13].

Each online peer p stores pointers to c well defined overlay neighbors (or contacts) which are specified by the individual DHT protocols. To maintain this 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. Our estimator monitors the changes in this neighbor list and collects different realizations of the random variables T_{on} and T_{off}. Thereby, $obs(i)$ is the ith observation made by the peer and $time(i)$ is the time when the observation was made. The observation history is stored in a list which contains up to k_{max} entries. Furthermore, a peer

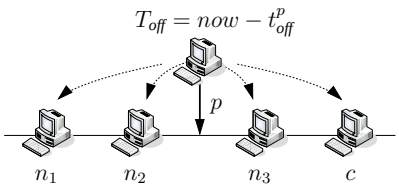

Fig. 1. Peer p rejoins the network and sends its offline duration to its c neighbors

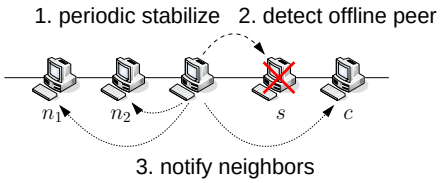

Fig. 2. Peer p only monitors its direct neighbor s but distributes its observations

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.

The shorter a peer stays offline on average the higher is the join rate. To obtain realizations of T_{off}, a peer stores the time t^p_{off} when it last went offline. The next time it goes online it calculates the duration of its offline session as $now - t^p_{off}$ and sends this value to its c overlay neighbors n_i (cf. Fig. 1). Note that the information can be piggybacked on other protocol messages to avoid unnecessary overhead. 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.

To obtain realizations of T_{on} we proceed as follows. In a DHT system, a peer p periodically contacts at least one neighbor s to stabilize the overlay structure (cf. Step 1 in Fig. 2). 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 Fig. 2), 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 Fig. 2. If the DHT applies some kind of *peer down alert mechanism* [1, 11], 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, e.g., true if p went online after s or if s became the successor of p due to churn. 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 duration $now - t^s_{on}$. 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.

When a peer joins the network, it first needs to obtain some observations before it can make a meaningful estimate of the churn rate. Therefore, we use the overlay network as a memory of already obtained observations. 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. An alternative is to invest more overhead by periodically contacting a number of peers instead of just one. Mahajan et al. [14] present an algorithm which relies on the fact that a peer continuously observes c overlay neighbors. Such a peer should on average observe one failure every

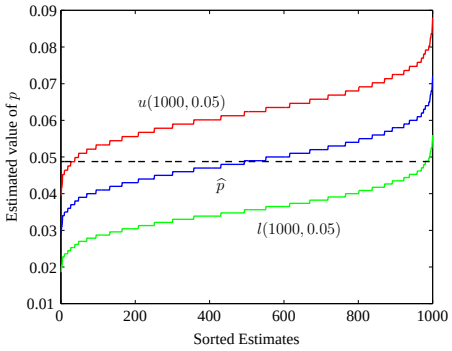

Fig. 3. Sorted estimates ($k = 10^3$, $\alpha = 95$)

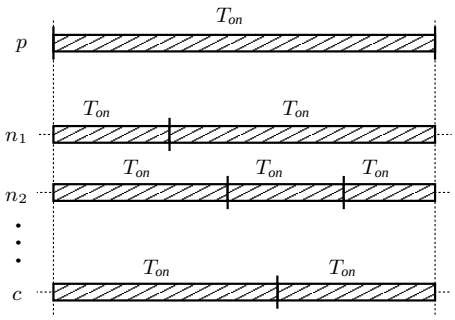

Fig. 4. Observations in lifetime of peer p

$\Delta t = \frac{1}{c} \cdot E[T_{on}]$. Thus, if a peer observes k failures in Δt the mean online time of a peer can be estimated as:

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

In addition to the periodic contact to c neighbors, the algorithm also has to struggle with the correctness of the neighbor pointers and the problem of obtaining enough observations during the lifetime of the peer.

3.2 Derivation of the Churn Rate

In this section we will use the following notation: For a random variable X, we denote $x(t)$ as the pdf, $X(t)$ as the cdf, and $E[X]$ as the mean. Estimated values will be marked using a hat. 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 can rely on robust estimates like the empirical mean and the empirical standard deviation.

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 corresponding 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 of T_{on} and T_{off} give a first idea about the churn in the system, the main purpose of the estimator is 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 p that this contact will stay online for less than t_{stab} seconds. An unbiased point estimator for this probability is given by:

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

where $|\cdot|$ indicates the cardinality of a set. The $100(1-\alpha)$ confidence interval for $\widehat{p}$ can be calculated using the following bounds:

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

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 be used as estimates themselves.

We simulated an overlay with $t_{stab} = 30\,\mathrm{s}$ where the online time of a peer was exponentially distributed with mean $E[T_{on}] = 600\,\mathrm{s}$. Under these conditions, the probability p that a specific peer goes offline before the next stabilization instant is 4.88%. Fig. 3 shows the sorted estimates of p and the corresponding upper and lower bounds from 1000 peers. 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 Eqn. (1) 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. If the type of distribution is known a priori, the peer can use the corresponding *Maximum Likelihood Estimator* (MLE) to estimate the parameters of the distribution. 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 [15] 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 [16] of the online distribution.

To show the importance of using the overlay network as a memory for already made observations, we regard the random variable X which describes the number of observations a peer makes during its lifetime provided that it continuously observes c overlay neighbors. This concept is visualized in Fig. 4 where we assume that a neighbor n_i which went offline is immediately replaced by another peer. The random variable X corresponds to the number of leave events in the figure and can be computed as

$$P(X=i) = \int_0^\infty t_{on}(t) \cdot P(X=i|T_{on}=t)\ dt. \quad (3)$$

In the case of exponentially distributed online times, this can be written as

$$P(X=i) = \int_0^\infty \lambda e^{-\lambda t} \cdot \frac{(c\lambda t)^i}{i!} \cdot e^{-c\lambda t}\, dt = \frac{c^i}{(c+1)^{i+1}}, \quad (4)$$

since the number of departures in a fixed interval of length t is Poisson distributed with parameter $c \cdot \lambda$.

To compare this theoretical approximation to practical values, we simulated an overlay network with $T_{on} = 300\,\mathrm{s}$, $t_{stab} = 30\,\mathrm{s}$, and $c = 40$, where the online/offline time of a peer is exponentially distributed. The maximum size of the history was set to $k_{max} = 100$. Fig. 5 shows the probability density function of X for both the analysis

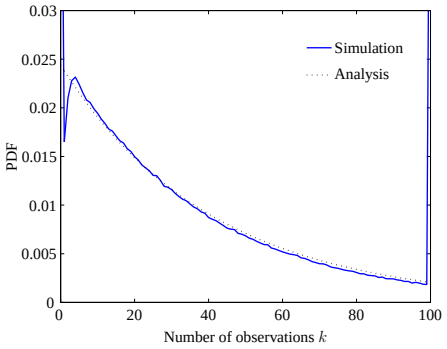

Fig. 5. Expected observations for $c = 40$

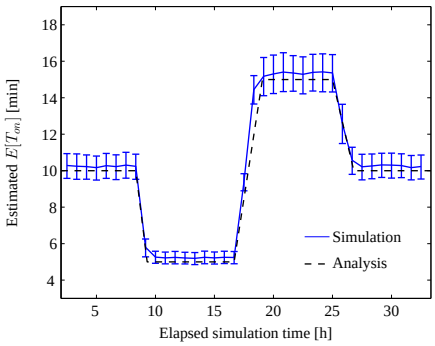

Fig. 6. Response to churn changes

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 s after the peer joined the network. Thus, all peers which stay online for less than 30 s, can never make an observation. In conclusion, both the analytical and the simulation results show that a peer does not make enough observations during its lifetime in order to derive a meaningful estimate and a good estimator should therefore utilize the overlay network as a memory for already made observations.

The more observations a peer makes per time unit, the faster it can react to changes in the global churn rate. 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. 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:

$$P\left(T_{obs}^{leave} < t\right) = 1 - P\left(T_{on} \geq t\right)^{c+1} = 1 - e^{-(c+1)\lambda t}. \tag{5}$$

If the distribution is not known, we can still easily compute the mean of T_{obs}^{join} as $E\left[T_{obs}^{join}\right] = \frac{E[T_{off}]}{c+1}$. The calculation is a little 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 $\epsilon_{on} = \frac{t_{stab}}{2}$ which leads to

$$E\left[T_{obs}^{leave}\right] = \frac{E[T_{on}] + \epsilon_{on}}{c+1} \tag{6}$$

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.

$$E[R] = E_{old}[T_{on}] + \frac{k_{max}}{c+1} \cdot (E_{new}[T_{on}] + \epsilon_{on}) \tag{7}$$

Fig. 6 compares the analytical response time to that obtained from a simulation run. In the simulation we again used exponentially distributed online/offline times, set $k_{max} = 100$, $c = 10$, $t_{stab} = 30\,\mathrm{s}$, and changed $E[T_{on}]$ from 10 min to 5 min to 15 min and back to 10 min after $8.33\,\mathrm{h}$, $16.66\,\mathrm{h}$, and $25\,\mathrm{h}$ 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. 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 the stabilization period of 30 s, the estimated values lie $\epsilon_{on} = 15\,\mathrm{s}$ above the actual value.

4 Numerical Results

In this section we will evaluate the proposed estimator using simulations. 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 $t_{stab} = 30\,\mathrm{s}$ and the size of neighbor list is $c = 20$. We will further assume that there are 40000 initial peers with $E[T_{on}] = E[T_{off}]$, 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 and T_{off} can be calculated in an analogous way.

4.1 Proof of Concept

The main purpose of this section is to show that the theoretic concept of the proposed estimator as described in Section 3 does work equally well in practice. We focus on Chord since it is the currently most studied DHT network architecture. Additionally, we will provide analytical calculations verified by simplified simulations, focusing on 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 $t_{stab} = 30\,\mathrm{s}$ 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 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.

In practice too high or too low estimates might have critical 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

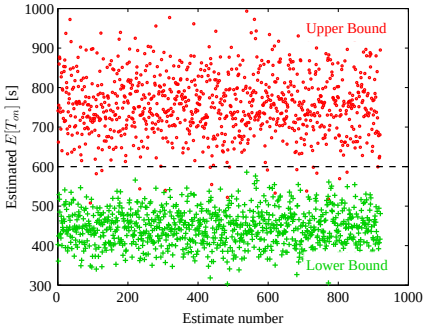

Fig. 7. Upper and lower confidence levels

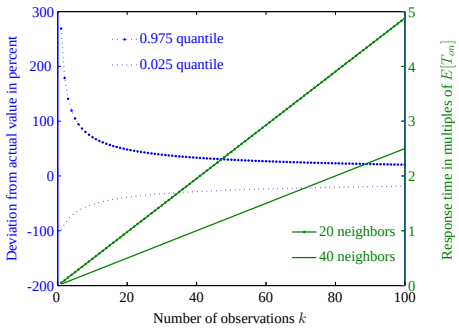

Fig. 8. Accuracy vs. responsiveness

value itself. Fig. 7 shows the upper and lower bounds of the 99% confidence interval for the mean. 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 inaccurate values.

4.2 Accuracy and Responsiveness

We now take a closer look at the trade-off between accuracy and responsiveness in dependence of the size of the history. To express accuracy, 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 Fig. 8 using the left y-axis. It can be recognized that increasing the history size results in more accurate estimates which decreases exponentially over 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 Eqn. (7). Responsiveness increases linearly with k (cf. green solid curves of Fig. 8 with right y-axis) and its slope 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. The study shows that depending on the application requirements, a trade-off can be made between higher accuracy and faster responsiveness by changing the number of considered observations.

In order to provide a more comprehensive study of the responsiveness of the estimator and to validate our analytical approximation in Eqn. (7), 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 the reaction to changes of the churn rate, see Fig. 9. For different churn rates of $E[T_{on}] = 300\,\mathrm{s}$, 600 s, and 900 s, the inter-observation time is shown over the number of overlay contacts. The dashed lines are the results obtained by the approximation, cf. Eqn. (6). It can be seen that the inter-observation time decreases exponentially and that the analytical curves match well with those obtained by simulations. A greater number than 20 neighbors is

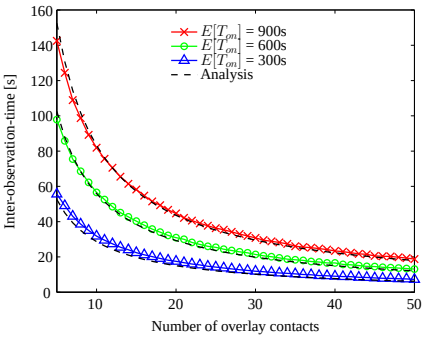

Fig. 9. Responsiveness to different churn

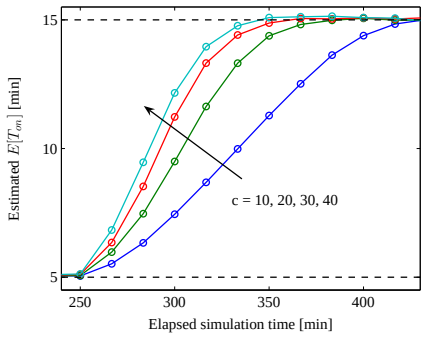

Fig. 10. Reaction to global churn changes

not justified due to the small improvement in responsiveness and the higher overhead in maintaining those neighbors. Smaller values of $E[T_{on}]$ result in smaller values of the inter-observation time, but 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 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 5min to 15min after a simulation time of 250 min. In Fig. 10 each data point shows the average of the estimated $E[T_{on}]$ values of all online peers at the same time instant. Again the more neighbors there are, the faster the estimator approaches the new churn rate. However, increasing the number of neighbors beyond $c = 20$ does not justify its additional overhead. Thus, using 20 overlay neighbors, as e.g. suggested in Kademlia, is a reasonable choice.

4.3 Practicability and Implementation Aspects

In practice, it is desirable that all peers obtain equal estimates in order to derive similar input parameters for the maintenance algorithms of the P2P network. However, those algorithms are performed between direct neighbors of the DHT. 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. Fig. 11 depicts the autocorrelation over the number of neighbors and shows that there is a high correlation among neighboring peers. The 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.

A possible application of the proposed estimator is self-tuning the stabilization of the overlay structure. In practice, the stabilization interval, i.e. the frequency at which

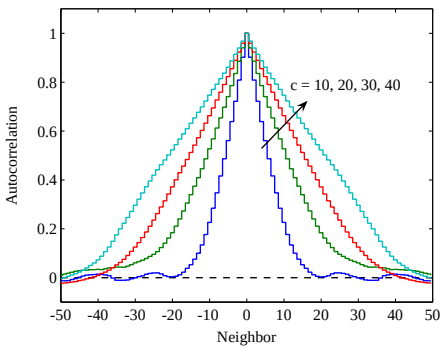

Fig. 11. Correlation among neighbor peers

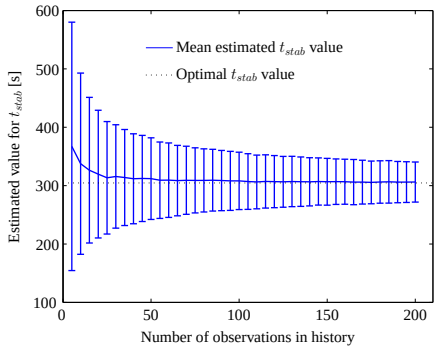

Fig. 12. Choice of stabilization interval

overlay neighbors are contacted to update the neighbor lists, is a fixed value. This results in unnecessary overhead when there is no churn in the network, but 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 t_{stab}, a peer should therefore estimate the current churn rate to derive the probability that the overlay structure becomes instable, i.e. that all neighbors will be offline before the next stabilization call. For example, given a mean online time of $E[T_{on}] = 600\,\mathrm{s}$, a peer needs to stabilize at least every 300 s in order to maintain the overlay stability with a probability of 99.99%. In Fig. 12, the mean and standard deviation of the t_{stab} derived from estimation is shown over the size of the observation history. 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.

5 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 and can 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 autonomously 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 input, and the insightful discussions during the course of this work.

References

1. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling Churn in a DHT. In: USENIX Annual Technical Conference, Boston, MA (2004)
2. Binzenhöfer, A., Staehle, D., Henjes, R.: On the Stability of Chord-based P2P Systems. In: GLOBECOM 2005, St. Louis, MO, USA (2005) 5
3. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In: Proc. of ACM SIGCOMM'01, San Diego, CA (2001)
4. Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Proc. of IPTPS'02, Cambridge, MA (2002)
5. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proc. of Middleware'01, Heidelberg, Germany (2001)
6. Liben-Nowell, D., Balakrishnan, H., Karger, D.: Analysis of the Evolution of Peer-to-Peer Systems. In: Proc. of ACM PODC, Monterey, CA (2002)
7. Krishnamurthy, S., El-Ansary, S., Aurell, E., Haridi, S.: A statistical theory of chord under churn. In: Proc. of IPTPS'05, Ithaca, NY (2005)
8. Godfrey, P.B., Shenker, S., Stoica, I.: Minimizing churn in distributed systems. In: Proc. of ACM SIGCOMM, Pisa, Italy (2006)
9. Gummadi, K.P., Dunn, R.J., Saroiu, S., Gribble, S.D., Levy, H.M., Zahorjan, J.: Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: Proc. of ACM SOSP'03, Bolton Landing, NY (2003)
10. Bhagwan, R., Savage, S., Voelker, G.: Understanding availability. In: Proc. of IPTPS'03), Berkeley, CA (2003)
11. Castro, M., Costa, M., Rowstron, A.: Performance and dependability of structured peer-to-peer overlays. In: Proc. of DSN'04, Washington, DC (2004)
12. Ghinita, G., Teo, Y.: An adaptive stabilization framework for distributed hash tables. In: Proc. of IEEE IPDPS, Rhodes Island, Greece (2006)
13. Binzenhöfer, A., Staehle, D., Henjes, R.: On the Fly Estimation of the Peer Population in a Chord-based P2P System. In: 19th International Teletraffic Congress (ITC19), Beijing, China (2005)
14. Mahajan, R., Castro, M., Rowstron, A.: Controlling the cost of reliability in peer-to-peer overlays. In: Proc. of IPTPS'03, Berkeley, CA (2003)
15. Stephens, M.A.: Edf statistics for goodness of fit and some comparisons. In: Journal of the American Statistical Association. Volume 69. (1974) 730–739
16. Chen, E.J., Kelton, W.D.: Quantile and histogram estimation. In: Proc. of 33nd Winter Simulation Conference, Washington, DC (2001)