University of Würzburg
Institute of Computer Science
Research Report Series

Improving the Performance and Robustness of Kademlia-based Overlay Networks

Andreas Binzenhöfer and Holger Schnabel

Report No. 405 April 2007

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

Improving the Performance and Robustness of Kademlia-based Overlay Networks

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

Abstract

Structured peer-to-peer (p2p) networks are highly distributed systems with a potential to support business applications. There are numerous different suggestions on how to implement such systems. However, before legal p2p systems can become mainstream they need to offer improved efficiency, robustness, and stability. While Chord is the most researched and best understood mechanism, the Kademlia algorithm is widely-used in deployed applications. There are still many open questions concerning the performance of the latter. In this paper we identify the main problems of Kademlia by large scale simulations and present modifications which help to avoid those problems. This way, we are able to significantly improve the performance and robustness of Kademlia-based applications, especially in times of churn and in unstable states. In particular, we show how to increase the stability of the overlay, make searches more efficient, and adapt the maintenance traffic to the current churn rate in a self-organizing way.

1 Introduction

There is both theoretical and practical evidence that p2p networks have a potential to support business applications. They are scalable to a large number of customers, robust against denial of service attacks, and do not suffer from a single point of failure. Skype [12], a p2p based VoIP application, e.g., serves millions of people every day. The main task of the underlying p2p network is to support efficient lookups for content stored in the overlay. The latest generation of p2p networks, the so called Distributed Hash Tables (DHTs), was especially designed to handle this task in a fast and scalable way. There are numerous different DHTs proposed in literature: CAN, Pastry, Chord, and Kademlia, to name just a few. All those algorithms do have in common that each participating peer gets a unique identifier using a hash function, while a distance metric is defined on these identifiers. In order to maintain the stability of the overlay each peer usually has a very good knowledge about its neighbors and some additional pointers to more distant peers used as shortcuts to guarantee fast lookups. In the research community Chord became the most studied algorithm in the last few years, which is possibly due to its easy to analyze ring structure. The scalability [10] [2], the behavior under churn [5] and the overlay stability of Chord [3] are well understood.

The majority of deployed overlay networks, however, make use of the Kademlia protocol [7]. It replaces the server in the latest eMule modifications and is used as a distributed tracker in the original BitTorrent as well as in the Azureus client [1]. The latter continuously attracts more than 800.000 simultaneous users world wide. Despite all this there are only few scientific papers evaluating the performance of the Kademlia algorithm. In [6] the performance of different DHT

algorithms including Kademlia is evaluated and compared. Modifications to support heterogeneous peers are introduced in [4]. Finally in [11] an analysis of the lookup performance of Kad, the Kademlia-based DHT used in eMule, is given. The authors examine the impact of routing table accuracy on efficiency and consistency of the lookup operation and propose adequate improvements.

In order to understand the performance of Kademlia in greater detail, we implemented a detailed discrete event simulator in ANSI-C based on the algorithm given in the original paper [7]. In particular, we studied the search duration, the overlay stability and the required maintenance traffic. In this paper we present the insights gained during our simulations. We will describe the weak points we discovered and pinpoint their root causes. For each problem we will present an optimization, which eliminates the disadvantages and makes Kademlia a protocol more feasible for business applications.

The remainder of the paper is structured as follows: In Section 2, we recapitulate the main aspects of the original Kademlia algorithm. A brief description of our simulator and the corresponding user model is given in Section 3. The discovered problems, their causes, and the solutions are summarized in Section 4. Section 5 finally concludes the paper.

2 Standard Kademlia

Kademlia is a DHT-based p2p mechanism which is used to efficiently locate information in an overlay network. A hash table is a data structure that associates keys with values. A distributed hash table (DHT) assigns the responsibility of parts of the value range of the hash function, i.e. of the address space S, to different peers. In order to retrieve the data, DHTs apply sophisticated routing schemes, such as self-balancing binary search trees. Each peer stores contact information about other peers in order to route query messages.

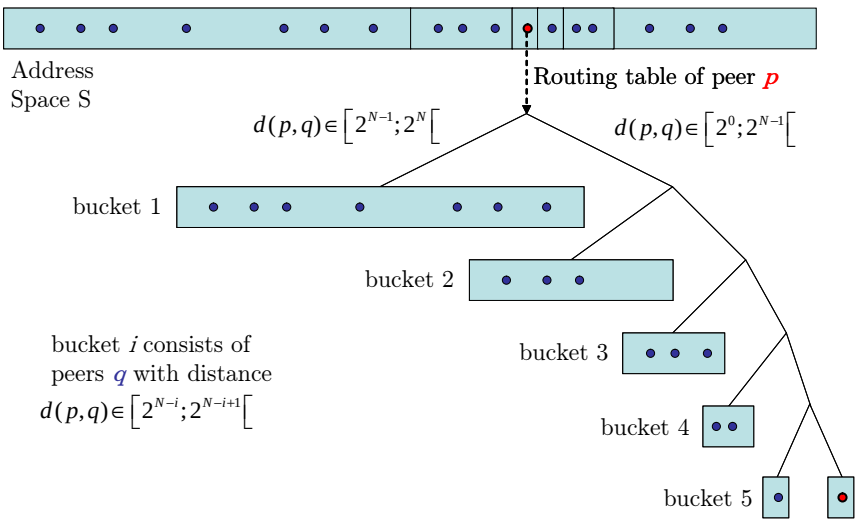

Figure 1: Routing table of peer p

In Kademlia, the branches of the binary search tree are represented as buckets, cf. Figure 1.

The collection of buckets form the routing table. Bucket i of peer p's routing table is a list of peers which have a certain distance to peer p. Kademlia uses 160-bit identifiers for the address space and applies the XOR metric, i.e.,

$$S = \{0;1\}^N \quad \text{with } N = 160 \tag{1}$$

$$\begin{aligned} d: \quad S \times S &\rightarrow [0; 2^N], \\ (p, q) &\mapsto p \oplus q. \end{aligned} \tag{2}$$

This means that bucket i in the routing table of peer p covers all peers q with distance $d(p, q) \in [2^{N-i}; 2^{N-i+1}[$, cf. Figure 1. In order to keep the size of the routing table small enough, a bucket has at most k entries and is also referred to as k-bucket. This results in a maximal number of routing table entries of $k \cdot N$. A more detailed description of the Kademlia algorithm can be found in [7].

3 Simulator Details

In order to evaluate the different performance aspects of Kademlia, we developed a discrete event simulator according to the algorithms in [7]. As stated above, for each $0 \leq i < 160$ a peer keeps a bucket of k peers of distance between 2^{N-i} and 2^{N-i+1} from itself according to the XOR metric. Thereby the routing table is adapted dynamically. That is, each peer starts with one single bucket covering the entire address space and recursively splits the bucket containing the peer's own ID as soon as this bucket holds more than k entries. This results in an up-to-date routing table reflecting the current state of the overlay network as shown in Figure 1. When many peers leave the system, Kademlia merges the corresponding buckets accordingly.

Furthermore, a peer is able to insert documents into the overlay network. To guarantee their availability, each of these documents is stored at the k closest peers to the document's ID. If the document was not received from another peer for T_{rep} minutes, the corresponding peer republishes the document, i.e. it sends the document to the remaining $k - 1$ peers of the replication group. When searching for a document a peer recursively sends parallel queries to the α closest peers it knows. The next recursion begins as soon as the peer received β answers. This guarantees that a searching peer will only run into a timeout if $\alpha - \beta + 1$ peers do not answer within one specific search step. If not stated otherwise, we use the default parameters $T_{rep} = 60$ minutes, $\alpha = 3, \beta = 2$, and $k = 20$.

To model end user behavior, we randomly chose join and leave events for each peer. To be comparable to other studies in literature a peer stays online and offline for an exponentially distributed time interval with a mean of E_{on} and E_{off} respectively. When online, the peer issues a search every E_{search} minutes, where the time between two searches is also exponentially distributed. Using different distributions mainly changes the quantitative but not the qualitative statements made during the remainder of this paper. To increase the credibility of our results [8], we include the 95 percent confidence intervals where appropriate.

4 Improvements

All structured p2p networks have been designed to scale to a large number of peers in the overlay. Therefore the real scalability issue of such systems is not in terms of system size but in terms of churn [9]. That is, the frequency at which peers join and leave the system has significantly more influence on its robustness and stability than the mere size of the system itself. In this section we uncover the problems caused by churn and show how to avoid them. In each simulation we use a total of 40000 peers, which we found to be sufficiently large to capture all important effects regarding the overlay size, and set $E_{on} = E_{off}$, resulting in an average overlay size of 20000 peers. The focus of our analysis of the simulation results is on qualitative behavior and not on quantitative statements.

4.1 Search Efficiency

The success and duration of a search for a document heavily depend on the correctness of a peer's pointers to other peers, i.e. on the correctness of the peer's routing table. In Kademlia the most crucial pointers are those to its k closest neighbors in the overlay. We measure the correctness of these pointers using two different variables:

- P_h: States how many of its current k closest neighbors a peer actually holds in its k-buckets.
- P_r: Represents the number of correct peers out of the k closest neighbors, which a peer actually returns when asked for.

Ideally a peer would not only know but also return all of its k neighbors.

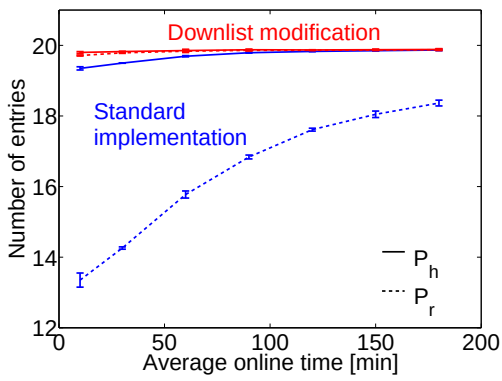

Figure 2: P_h and P_r in dependence of the churn rate

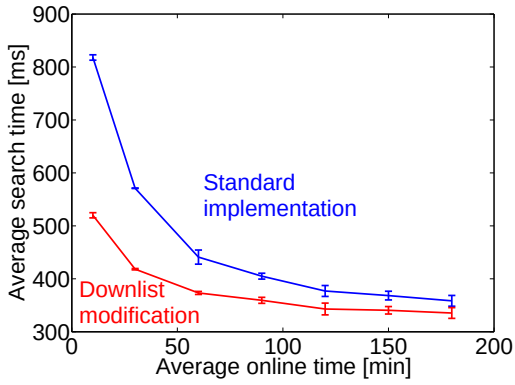

Figure 3: Influence of the downlist modification on the search efficiency

However, our simulations show that the standard implementation of Kademlia has problems with P_r. We set $k = 20$ and simulated the above described network for different churn rates. Figure 2 illustrates P_h and P_r in dependence of the churn rate. The mean online/offline time of a peer was variied between 10 and 180 minutes. Even though on average a peer knows almost

all its neighbors (P_h close to 20), it returns significantly less valid entries when queried (P_r as low as 13). The shorter a peer stays online on average, the less valid peers are returned during a search. The problem can be tracked down to the fact that there are still many pointers to offline peers in the corresponding k-bucket of the peer. The reason is that there is no effective mechanism to get rid of out-dated k-bucket entries. Offline entries are only eliminated (or moved to the cache) if a peer runs into a timeout while trying to contact an offline peer. A peer which identifies an offline node, however, keeps that information to itself. Thus, it is not unlikely that a node returns offline contacts as it has very limited possibilities to detect offline nodes. As a result more timeouts occur and searches take longer than necessary. Another problem is that searches are also getting more inaccurate, which has negative effects not only on the success of a search but also on the redundancy of the stored documents. The reason is that due to the incorrect search results documents will be republished to less than k peers or to the wrong peers.

4.1.1 Solution - Downlists

The primary reason for the above mentioned problem is that so far only searching peers are able to detect offline nodes. The main idea of our solution to this problem is that a searching peer, which discovers offline entries while performing a search, should share this information with appropriate other peers. To do so, a peer maintains a downlist consisting of all peers which it discovered to be offline during its last search. At the end of the search the corresponding entries of this downlist are sent to all peers which gave those entries to the searching peer during its search. These peers then also remove the received offline entries from their own k-buckets. This mechanism helps to get rid of offline entries by propagating locally gained information to where it is needed. With each search offline nodes will be eliminated.

The improved stability of the overlay is obviously bought by the additional bandwidth needed to send the downlists. From a logical point of view, however, it does require more overhead to keep the overlay stable under higher churn rates. In this sense, the additional overhead traffic caused by sending downlists is self-organizing as it automatically adapts to the current churn rate. The more churn there is in the system, the more downlists are sent.

It should also be mentioned, that without appropriate security arrangements a sophisticated attacker could misuse the downlist algorithm to exclude a target node by claiming in its downlist that this specific node had gone offline. However, this problem can be minimized by only removing those nodes which were actually given to the searching node during a search or additionally by verifying the offline status using a ping message. One could also apply trust or reputation based mechanism to exclude malicious nodes.

4.1.2 Effect on Search Efficiency

To compare the downlist modification to the standard implementation we again simulated a scenario with 20000 peers on average and calculated the 95 percent confidence intervals. Figure 2 proves, that the downlist modification has the desired effect on P_r, the number of correctly returned neighbors. Using downlists both P_h and P_r stay close to the desired value of 20, almost independent of the current churn rate. That is, even in times of high churn the stability of the overlay can be guaranteed.

This improved correctness of the overlay stability also has a positive influence on the search efficiency. In Figure 3 we plot the average duration of a search against the average online/offline time of a peer. In this context an overlay hop was modeled using an exponentially distributed random variable with a mean of 80 ms. Both curves show the same general behavior. The longer a peer stays online on average, the shorter is the duration of a search. However, especially in times of high churn, the downlist modification (lower curve) significantly outperforms the standard implementation. The main reason is that on average a peer runs into more timeouts using the standard implementation, as it queries more offline peers during a search. The effects on the maintenance overhead will be discussed in Section 4.3.

4.2 Overlay stability

When peers join and leave the overlay network, the neighbor pointers of a peer have to be updated accordingly. As mentioned above, the downlist modification greatly improves the correctness of the k closest neighbors of a peer. To understand this effect in more detail, we have a closer look at a single simulation run. We consider a mean online/offline time of 60 minutes and an average of 20000 peers for both the standard implementation and the downlist modification.

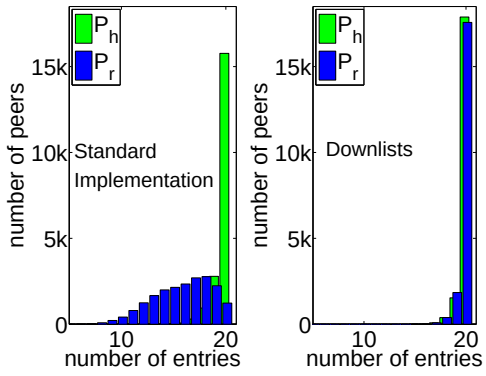

Figure 4: P_h and P_r for the standard implementation and the downlist modification

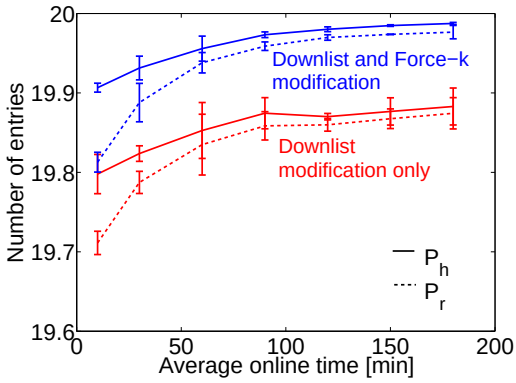

Figure 5: Effect of Force-k under churn

Figure 4 illustrates the distribution of P_h and P_r in both scenarios. As can be seen in the left part of the figure, almost all peers know more than 17 of their 20 closest neighbors using the standard implementation. However, the number of correctly returned peers P_r is significantly smaller for most peers. This problem is greatly reduced by the downlist modification as can be seen in the right part of the figure. In this case, the number of known and the number of returned peers are almost equal to each other. Yet, there are still some peers, which do not know all of their 20 closest neighbors. This is in part due to the churn in the overlay network. However, simulations without churn produce results, which are comparable to those shown in the right part of Figure 4. The cause of this problem can be summarized as follows: Let B_p be the k-bucket of peer p, which includes the ID of peer p itself and $B_{\bar{p}}$ the brother of B_p in the binary tree whose leaves represent the k-buckets as shown in Figure 6. Then according to the Kademlia

algorithm bucket B_p is the only bucket which will be split. However, if only $e < k$ of the actual k closest contacts fall into this bucket, then $v = k - e$ of these contacts theoretically belong into its brother $B_{\bar{p}}$.

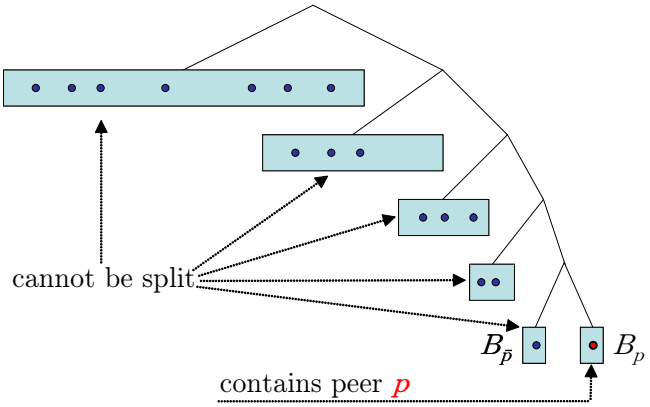

Figure 6: B_p and its brother $B_{\bar{p}}$ in the Kademlia routing table

Now, if this bucket is full it cannot be split. Thus, if some of the v contacts are not already in the bucket, it is very unlikely that the peer will insert them into its buckets. The reason is, that a new contact will be dropped in case the least recently seen entry of $B_{\bar{p}}$ responds to a ping message. Since in a scenario without churn all peers always answer to ping messages, new contacts will never be inserted into $B_{\bar{p}}$, even though they might be among the k closest neighbors of the peer. In the original paper it is suggested to split additional buckets in which the peer's own ID does not reside in order to avoid this problem. However, this has two major drawbacks. At first, it is a very complex process, which is vulnerable to implementation errors. Secondly, it involves a great deal of additional overhead caused by bucket refreshes and so on and so forth. In the next section, we therefore develop a simple solution, which does not require any additional overhead.

4.2.1 Solution - Force-k

As stated above, it is possible, that a peer does not know all of its k closest neighbors, even in times of no churn. To solve this problem, we need to find a way to force a peer to always accept peers belonging into $B_{\bar{p}}$ in case they are amongst its k closest neighbors. Suppose a node receives a new contact, which is among its k closest neighbors and which fits into the already full bucket $B_{\bar{p}}$. So far, the new contact would have been dropped in case the least recently seen entry of $B_{\bar{p}}$ responded to a ping message. Compared to this, the Force-k modification ensures that such a contact will automatically be inserted into the bucket. In order to decide which of the old contacts will be replaced, one could keep sending ping messages and remove the first peer, which does not respond. However, this again involves additional overhead in terms of bandwidth. A faster and passive way is to put all entries of $B_{\bar{p}}$, which are not among the k closest peers into a list l and drop the peer which is the least useful. This could be the peer which is most likely to be offline or the peer which has the greatest distance according to the

XOR metric.

In our implementation, we decided to consider a mixture of both factors. Each of the entries e of list l is assigned a specific score

$$s_e = t_e + d_e \tag{3}$$

and the one with the highest score will be dropped. Thereby, t_e is intended to be a measure for the likelihood of peer e to be offline and d_e for the distance of peer e to peer p. The exact values of t_e and d_e are obtained by taking the index of the position of the corresponding peer in the list, as if it was sorted ascending by the time least recently seen or by the peer's distance respectively. That is, if e is the least recently seen peer ($t_e = 1$) and has the third closest distance to peer p ($d_e = 3$) it is assigned a score of $s_e = 4$.

4.2.2 Effect on Stability

We investigated the impact of the Force-k modification on the stability of the overlay network in various simulations. In scenarios without churn, all peers finally know and return all of their k closest neighbors. The corresponding figures show lines parallel to the x-axis at a value of $k = 20$. It is therefore more interesting to regard the overlay stability during churn phases.

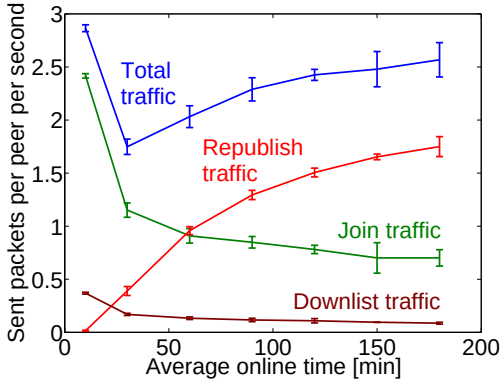

Figure 7: The maintenance traffic of a peer split into its components

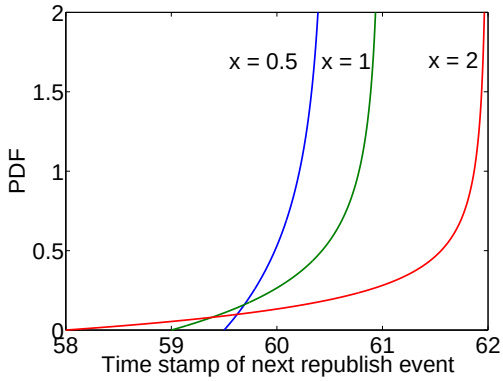

Figure 8: PDF of I_{rep} for different values of x

In Figure 5, we plot the average online time of a peer against the number of known and returned neighbors using the same simulation scenario as before. The two lower curves correspond to our previous results using the downlist modification. The two upper curves represent the Force-k modification in combination with the downlist modification. It can be seen that the Force-k algorithm also improves the stability of the overlay in times of churn. While the appearance of the curves is similar, there are more neighbors known (solid lines) and returned (dashed lines) as compared to using only the downlist modification. Even if a peer stays online for only 10 minutes on average, it will know about 19.9 out of 20 neighbors and return more than 19.8 correct entries. By improving the correctness of the neighbors, the Force-k modification also increases the search success rate and the redundancy of stored documents.

4.3 Redundancy Overhead

The bandwidth required to maintain a stable overlay and to ensure the persistence of stored documents directly reflects the costs for a peer to participate in the network. We simulated a network with 20000 peers on average and recorded the average number of packets per second sent by a peer while it was online. Figure 7 illustrates the average traffic per peer in dependence of the average online time of a peer. In addition to the total traffic, the figure also shows its three main components, the join, the republish, and the downlist traffic.

Since E_{search}, the average time between two searches of a peer, was set to 15 minutes, the search traffic per peer per second can be neglected in this scenario and is thus not shown in the figure. The same is true for the traffic caused by bucket refreshes, since a specific bucket is only refreshed if it has not been used for an entire hour. The Force-k algorithm is performed locally and does thus also not produce any additional overhead.

It can be seen in the figure that the downlist traffic automatically adapts itself to the current churn rate. The more frequently the peers join and leave the system, the more downlist traffic is produced by a peer on average. In general, the small amount of bandwidth needed to distribute the downlists is also easily compensated by the improved stability of the overlay. The major part of the traffic is caused when joining the network and republishing documents. It is obvious that the average amount of join traffic increases if a peer stays online for a shorter period of time. The join traffic cannot and should not be avoided as it is necessary for a peer to make itself known when it joins the network. Moreover, the join traffic already shows a self-organizing behavior. The more churn there is in the system, the more joins there are in total and the more overhead is produced to compensate the problems caused by the churn.

At first, the run of the curve representing the republish traffic seems to be counter-intuitive. The less churn there is in the system, the more republish traffic is sent by a peer on average. However, the reason becomes obvious, if one takes into account that the longer a peer stays online on average, the more likely it gets that there are republish events. In fact, the probability that a peer stays online for longer than 60 minutes given the corresponding average online time E_{on}, resembles the run of the republish curve. The reason why the total amount of republish traffic exceeds the remaining traffic so significantly is as follows: Each document is stored at the k closest nodes to its ID, the so called replication group. To compensate for nodes leaving the network, each peer sends the document to all other peers of the replication group if it has not received the document from any other peer for $T_{rep} = 60$ minutes. The idea behind this republish mechanism is that one peer republishes the document and all other peers reset their republish timers accordingly. Since the republishing peer sends the document to all peers of the replication group simultaneously, the peers reset their timers at approximately the same time. The next time the first peer starts to republish the document, it has to search for the corresponding replication group before it can redistribute the document. However, during this search the republish timers of the other peers are likely to run out and they will start to republish the document as well. For this reason, a document might get republished by up to k peers instead of just one single peer, resulting in unnecessary overhead traffic. This problem of synchronization is already mentioned in the original paper. In the following section, we present a solution, which greatly reduces the republish overhead and which is also resistant against churn.

4.3.1 Solution - Betarepublish

The synchronization problem of the republish process arises if all peers of a replication group have approximately the same time stamp for the next republish event. At first this seems to be unlikely. However, each time a peer republishes a document all other peers of the replication group receive this document at approximately the same time and are thus synchronized again. The main idea to avoid this problem is to assure that all peers use different time stamps. To achieve this, each peer chooses its time stamp randomly in the interval $[T_{rep} - x, T_{rep} + x]$ instead of exactly after $T_{rep} = 60$ minutes. Let I_{rep} be the random variable describing the time stamp of the next republish event. Then we want I_{rep} to be distributed in such a way, that only few peers start republishing at the beginning of the interval and the probability to republish increases towards the end of the interval. This can, e.g., be achieved by setting:

$$I_{rep} = (T_{rep} - x) + 2 \cdot x \cdot I_{beta} \tag{4}$$

where I_{beta} is a random variable with density

$$i_{beta}(t) = \begin{cases} \frac{t}{\sqrt{(1-t)} \cdot B(2,0.5)} & \text{if } 0 < t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

and $B(\alpha, \beta)$ is the beta function, defined by

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1} (1-t)^{\beta-1} \, \mathrm{d}t \tag{6}$$

Thereby x should be small compared to T_{rep} but still significantly larger than the duration of a search. Figure 8 shows the probability density function of I_{rep} for different values of x. All peers will set their time stamps somewhere in the interval $[60 - x, 60 + x]$. The probability for a peer to set its time stamp is still very low at the beginning of the interval. It then ascends significantly towards the end of the interval. In the case of $T_{rep} = 60$ minutes, $x = 2$ minutes is a reasonable choice, since it offers a long period of time with a low probability of republish events. This way, the republish traffic will be significantly reduced as it becomes very likely that only one or a few peers actually start a republish process. Again, note that a peer does only republish a document if it has not received it from another peer for $T_{rep} = 60$ minutes.

4.3.2 Effect on Overhead

In this section we will have a look at the influence of the Betarepublish modification on the average amount of republish traffic sent by a peer.

Figure 9 shows the average number of republish packets per peer per second in dependence of the average online time. We compare the results for simulations using the standard implementation, our two previous modifications, and all modifications including Betarepublish. First of all the average republish traffic of a peer is increased by using the downlist modification. The reason is that using the standard implementation there are more offline nodes in the k-buckets during times of churn. Thus documents are republished to less peers, which reduces the republish traffic but also the redundancy in the system. The additional traffic introduced by the downlist modification is therefore used to improve the availability of documents.

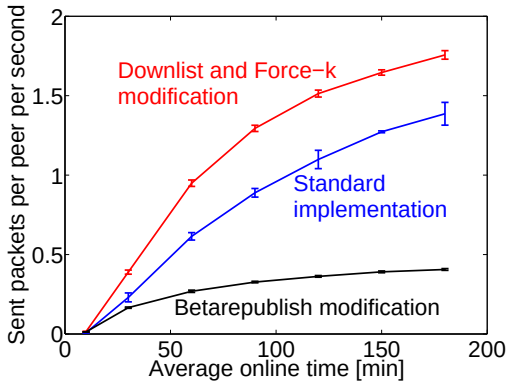

Figure 9: Maintenance traffic caused by republish processes

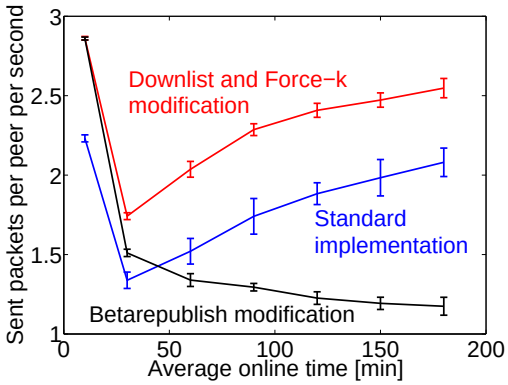

Figure 10: Total maintenance traffic in dependence of the churn rate

The Betarepublish modification is applied in an effort to minimize the traffic which is necessary to achieve this availability. The figure shows that Betarepublish indeed reduces the amount of required republish traffic significantly. The Betarepublish traffic lies well beneath the standard implementation and also rises slower with an increasing average online time. Note that the Betarepublish modification does only avoid redundant traffic. It is still able to guarantee the same redundancy, stability, and functionality. Figure 10 shows how the reduced republish traffic influences the total traffic for the three regarded versions of Kademlia (Standard, downlists and Force-k, all modifications). At first, it can be seen that the use of downlists increases the total traffic as compared to the standard implementation. Again, this is desired overhead as it greatly helps to increase the robustness, the stability, and the redundancy of the overlay in an autonomous way.

By adding the Betarepublish modification, the total traffic is significantly reduced and no longer dominated by the republish traffic. While the average maintenance traffic sent by a peer in the standard implementation actually increases when there is less movement in the overlay network, it finally shows a self-organizing behavior when using all modifications. The less churn there is in the system, the less maintenance traffic is generated to keep the overlay network up to date. That is, the amount of bandwidth invested to keep the overlay running automatically adapts itself to the current conditions in the overlay.

5 Conclusion

In this paper we investigated the performance of the Kademlia protocol using a detailed discrete event simulator. We were able to detect and pinpoint some weak points regarding the stability and the efficiency of the overlay network. In this context, three modifications have been proposed to enhance the performance, the redundancy, and the robustness of Kademlia-based networks. With the help of downlists, the correctness of the neighbor pointers and the duration of a search is greatly improved. The Force-k modification ensures that a peer has a very good knowledge of its direct neighborhood, which greatly increases the stability as well as the overall performance. We

also introduced a new republish algorithm, which significantly reduces the total traffic needed to keep the overlay running. The improved version of Kademlia shows a self-organizing behavior as the amount of generated maintenance traffic autonomously adapts to the current churn rate in the system.

The proposed modifications can be used to support large scale p2p applications, which are able to sustain dynamic user behavior. Even though the algorithms have been introduced using Kademlia, they are by no means restricted to this protocol. Especially the downlist and the Betarepublish mechanisms can easily be applied to other DHTs like Pastry, CAN, or Chord.

Acknowledgements

The authors would like to thank Robert Henjes, Tobias Hofeld, and Phuoc Tran-Gia for the insightful discussions as well as the reviewers for their valuable suggestions.

References

[1] Azureus. URL: http://azureus.sourceforge.net/.

[2] A. Binzenhöfer and P. Tran-Gia. Delay Analysis of a Chord-based Peer-to-Peer File-Sharing System. In *ATNAC 2004*, Sydney, Australia, December 2004.

[3] Andreas Binzenhöfer, Dirk Staehle, and Robert Henjes. On the Stability of Chord-based P2P Systems. In *GLOBECOM 2005*, page 5, St. Louis, MO, USA, November 2005.

[4] Youki Kadobayashi. Achieving Heterogeneity and Fairness in Kademlia. In *Proceedings of IEEE/IPSJ International Workshop on Peer-to-Peer Internetworking co-located with Symposium on Applications and the Internet (SAINT2004)*, pages 546–551, January 2004.

[5] Supriya Krishnamurthy, Sameh El-Ansary, Erik Aurell, and Seif Haridi. A Statistical Theory of Chord under Churn. In *4th International Workshop on Peer-To-Peer Systems*, Ithaca, New York, USA, February 2005.

[6] Jinyang Li, Jeremy Stribling, Thomer M. Gil, Robert Morris, and M. Frans Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS04)*, San Diego, CA, February 2004.

[7] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS 2002*, Cambridge, MA, USA, March 2002.

[8] K. Pawlikowski, H.-D.J. Jeong, and J.-S. Ruth Lee. On credibility of simulation studies of telecommunication networks. In *IEEE Communications Magazine*, January 2002.

[9] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling Churn in a DHT. In *2004 USENIX Annual Technical Conference*, Boston, MA, June 2004.

[10] Ion Stoica, Robert Morris, David Karger, M. Frans. Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *ACM SIGCOMM 2001*, San Diego, CA, August 2001.

[11] Daniel Stutzbach and Reza Rejaie. Improving lookup performance over a widely-deployed dht. In *IEEE INFOCOM 2006*, Barcelona, Spain, April 2006.

[12] Skype Technologies. Skype. URL: http://www.skype.com.