Fast but economical: A simulative comparison of structured peer-to-peer systems

Ingmar Baumgart and Bernhard Heep
Institute of Telematics, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
Email: {baumgart,heep}@kit.edu

***Abstract*—In the past many proposals for structured peer-to-peer protocols have been published. They differ in properties like overlay topology and routing table maintenance. Furthermore, each protocol exhibits various parameters e.g. to adjust the size of the routing table or stabilization intervals, making it difficult to choose an optimal protocol and parameter set for a given scenario (e.g. churn rate, number of nodes). For this purpose, we developed the overlay simulation framework OverSim and implemented six well known structured overlay protocols. In this paper we first compare these protocols among each other. Furthermore, we study several recursive and iterative routing variants and show the effect of routing table redundancy and lookup parallelism on routing latency and bandwidth costs. For each overlay protocol we identify an optimal parameter set for a typical peer-to-peer scenario. Finally, we show how overlay protocols adapt to variations in churn rate and network size. Our results show considerable advantages of the protocols Kademlia and Bamboo, while De Bruijn based protocols reveal a lack of stability under churn.**

I. Introduction

Structured P2P systems enable the fast and cost efficient deployment of new services. Now that such P2P applications get actually deployed in the wild, the analysis of these overlay protocols gets even more important. Many different structured P2P systems —often denoted as *Distributed Hash Tables* (DHTs)— have been proposed. These protocols' basic service for applications is a so called *Key-based Routing* service (KBR). KBR protocols differ in several properties like overlay topology and routing table maintenance. Additionally, each protocol has several tunable parameters e.g. for maintenance intervals or routing table sizes. To make it worse most published evaluation results for these protocols are based on individual protocol implementations, making it difficult to compare the protocols' different structures or routing table maintenance strategies. With *OverSim* [1], we developed a simulation framework that helps to unify the behavior of KBR protocol implementations by extracting their common functions and put them into shared components and classes (e.g. routing, retransmission handling, bootstrapping) [2].

The contribution of this paper is as follows: We present a comprehensive comparison of several state of the art structured KBR protocols using our unified simulation framework OverSim in order to achieve a fair comparison. By means of simulations we are able to identify the optimal parameters of these protocols in a realistic P2P scenario. Afterwards, to evaluate scalability and reliability, we compare delivery ratio, bandwidth consumption and lookup latency of the mentioned protocols in different churn scenarios with varying numbers of nodes using the previously identified parameters. The simulation framework OverSim is published as an open-source project, thus the presented results can easily be reproduced and new P2P systems can be compared to today's common protocols with low effort.

The rest of the paper is organized as follows: In Section II we give a survey of related work covering analytical and simulative comparison of structured P2P systems. Section III focuses on implementation details of the covered KBR protocols. The utilized performance vs. cost evaluation framework (PVC) and the settings of the simulations are summed up in Section IV. The results are discussed in Section V and the paper ends with our conclusion in Section VI.

II. Related Work

There is a large number of publications (e.g. [3]–[7]) which analyze and compare various overlay topologies. But all these publications *neglect the effects of churn* and the complex behavior of the involved *stabilization mechanisms*. Rhea et al. [8] compared the two protocols *Chord* and *Bamboo* in scenarios with churn using a local testbed with a network emulator. The most comprehensive simulation study in networks with churn so far by Li et al. [9] includes the protocols *Kademlia, Kelips, Chord, Tapestry* and *OneHop*. In contrast to previous work the authors consider the mutual influence of overlay protocol parameters. This study is extended by Damm et al. [10] by an evaluation of the *Viceroy* protocol.

Our work addresses several limitations of these studies: First, the popular protocols *Pastry*, *Bamboo*, *Broose* and *Koorde* have not been considered in [9]. To our knowledge we are the first that evaluated the more recent constant node degree protocols *Broose* and *Koorde* in networks with churn. Second, in our work the communication costs for overlay signaling traffic were measured exactly on the basis of complete protocol implementations[1] – in [9] communication costs were only estimated according to the number of included IP addresses and nodeIds per signaling message. Compared to [9] we also used a more detailed underlay model which considers the delay of signaling messages due to queuing effects in the sender queues. Finally, our studies were done with 10,000 overlay nodes in contrast to only 1024 nodes in [9]. Using only 1024 nodes can hide several stabilization problems when using typical protocol parameters[2].

[1]In addition to simulations the protocol implementations were also verified in Planet-Lab experiments.

[2]E.g. even with a completely broken finger table, Chord still allows to reach all nodes by only following the successor lists if the network is sufficiently small.

III. Key-based Routing Protocols

The P2P protocols we compare in this paper all provide a *key-based routing service (KBR)* [2] to the application. To evaluate the various KBR protocols, we implemented them for our overlay simulation framework *OverSim* [1]. In the following, only remarkable features of these protocol implementation are described. The protocols' basic operation and different variants covered in this paper are specified in the corresponding publications.

A. Chord

We extended Chord [11] by the *aggressive join mode*, where the receiver of a JOIN call

- sets his predecessor pointer to the joining node
- puts his old predecessor as a hint in the response
- sends a *new successor hint message* to his old predecessor that updates his successor list

The joining node immediately sets his predecessor according to the hint in the join response message.

To provide redundancy in the finger table, an extended finger table for Chord was implemented as proposed in [11]. Here, a node that receives a *fix fingers call* returns his complete or part of its successor table. These nodes are probed and put into the call initiator's finger table in addition to the responding node. The next time a message has to be routed to this finger entry, the candidates' nodeIds as well as their proximity are considered for the next hop selection (*Proximity Routing (PR)* [12]).

B. Koorde

OverSim's implementation of Koorde corresponds to the original proposal in [13].

C. Pastry

The implementation of Pastry supports the original protocol [14] and the new version [15]. Node failures are detected by missed acknowledgements. These acks are sent by all nodes on the routing path to the last hop. A neighbor cache is utilized to reduce probe traffic and, in contrast to the original proposal, iterative lookups are supported.

D. Bamboo

For the calculation of RPC timeouts in Bamboo [8] [16], the usage of network coordinate systems is proposed. For a better comparability to other KBR protocols, this was deactivated for our simulations.

E. Kademlia

To prevent a loss of information about the k closest nodes to the current node, bucket splitting in a special way is proposed in [17]. In our implementation a special k-bucket, the *sibling table*, holds the s closest nodes with respect to the XOR metric. This way, all s closest nodes (= *siblings*) are kept in one bucket. Instead of the exhaustive iterative lookup procedure of the original Kademlia protocol, we introduce a *simple iterative* mode where lookups terminate when a node asked for close nodes to a destination key x knows all the k closest nodes to x (including the asked node itself). Additionally, a new parameter r is used to determine the number of returned nodes in a FIND_NODE response message independently of k (usually with $r < k$).

F. Broose

A node in a Broose [18] network needs to determine the network size. In OverSim, a node that wants to join the network sends 2^b B_BUCKET_REQUESTS via its bootstrap node to different destination keys in order to receive the *B-buckets* of the responsible nodes. The nodes from the received *B-buckets* are used to fill up the joining node's *R-buckets*. Then L_BUCKETS_REQUESTS are sent to the new *R-bucket* entries. With all the collected nodes, the network size can be estimated.

IV. Evaluation Method

The evaluation methodology of this paper is based on the *performance vs. cost evaluation framework (PVC)* [9]. There, two challenges in evaluating structured P2P systems are addressed:

- How can you quantify cost and performance for protocols that are tunable by means of more aggressive routing table maintenance, parallel lookups, more intensive searching for low latency neighbors, etc.?
- How can a parameter's impact on efficiency be judged?

PVC uses two metrics to define the performance of structured P2P systems: the average latency of successful routing procedures and their failure rate. To combine these metrics, failed routing attempts are counted as successful with a latency that equates to the routing timeout (10 seconds in our case). PVC simulates KBR protocols with different parameter combinations. The results are plotted with average bandwidth usage on the x-axis and median latency on the y-axis. Each data point in a plot represents a dedicated parameter combination. Two types of *convex hulls* can be sketched into a plot: The *overall convex hull* to determine the most efficient parameter combinations and the *parameter convex hull* to evaluate whether a particular parameter is more important than others according to the performance/cost tradeoff. For the overall convex hull, all parameter combinations are considered, for parameter convex hulls, the parameter of interest is fixed and all the others are varied. Data points inside the convex hulls represent suboptimal parameter combinations and are omitted in the following plots to increase readability.

In the following, we describe our simulation setup: All simulations were done with our overlay framework *OverSim* [1]. For the underlay we have chosen the SimpleUnderlayModel, which provides typical Internet latencies and supports modeling of queuing effects. For the simulation of churn, nodes are assigned a Weibull distributed session time with shape $k = 0.5$ and a mean lifetime and deadtime of $10,000$ seconds. This resembles the churn behavior that has been observed in the KAD file-sharing network [19] [20]. In Chapter V-D the mean lifetime (and deadtime) is varied between 100 and 100,000 seconds. On each node a test application periodically performs a lookup on a random nodeId of currently alive nodes using a truncated normally distributed interval with a mean of 60 seconds. Uniformly distributed 160 bit nodeIds were used for all protocols.

Each protocol is evaluated with different parameter combinations in a network of 10,000 nodes, each with 10 MBit/s access bandwidth we consider as typical medium access bandwidth of hosts in the Internet nowadays. Jitter with a variance of 10% of the unchanged coordinate-based delay is applied. The different routing modes are compared using Chord and Kademlia. Churn rates are varied using the optimal parameter settings from the simulations before. This is done with 10,000 simulated nodes. The protocols were finally tested on scalability using different network sizes between 10 and 20,000 nodes.

V. Results

In the following, we present the simulation results for the KBR protocols described above. For each protocol, a table of varied parameter values is attached, where the parameter combination is highlighted that is used for further evaluations (s. Sect. V-B, V-C, V-D, and V-E). All following plots show averaged values of multiple runs with 99% confidence intervals. Some results are presented without plots due to the lack of space.

A. Choosing optimal protocol parameters

1) Chord: The size of the successor list plays only a minor role in achieved routings latencies (figure omitted). It is obvious that a successor list size with more than two entries is needed for a stable network. Larger successor lists increase redundancy for routing decisions under churn. No significant difference in performance is noticeable between a successor list size of 8 and 16, except for parameter combinations with higher bandwidth consumption: From 250 bytes/s upwards, lower latencies are achieved with a successor list size of 16.

Fig. 1(a) shows the results with varying stabilize intervals. As varying the stabilization interval only affects nodes in the successor list, no remarkable effect on routing latencies can be observed, due to the fact that the successor list is usually used for the last hop only. Hence, a fixed value of 30 seconds is preferable, due to less maintenance traffic needed. In Fig. 1(b) the fix fingers interval was altered, which turned out to be an important parameter for Chord. A value of 240 seconds is appropriate, as smaller values lead to more traffic with no significant impact on lookup latency.

For the check predecessor interval, a value of 30 seconds appears to be a reasonable value, as no significant difference to shorter intervals can be recognized (figure omitted). The effect of enabled proximity routing with different values of alternative fingers is illustrated in Fig. 1(c). Basically, about half as high latencies can be achieved by using PR. Higher values than 8 for the number of alternative fingers do not significantly lead to lower latencies as in churn scenarios the last entries in successor tables are often outdated.

TABLE I: Parameter values for Chord

Parameter	Value
Successor list size	4, 8, **16**, 32
Stabilization interval	10 s, **30 s**, 90 s
Fix fingers interval	30 s, 120 s, **240 s**, 480 s
Check predecessor interval	10 s, **30 s**, 60 s
Size of extended finger table	0, 1, **4**, 8, 16

2) Koorde: Fig. 2 shows the results for Koorde. In our experiments the size of the De Bruijn list was set equal to the size of the successor list, since the list of De Bruijn nodes is built by using the successor list of the node preceding the De Bruijn pointer. The effect of the De Bruijn and successor list sizes is shown in Fig. 2(a). Using larger lists than $succ = 16$ nodes leads to bad performance, since then most entries are invalid. This is due to the fact that Koorde needs up to $succ$ stabilize intervals to refresh the complete successor list. Fig. 2(b) shows that the stabilize interval is more important for Koorde as for Chord, since the successor and De Bruijn lists are the main routing table. In Chord the routing mainly involves the finger table and the successor list is only used in the last hop.

As shown in Fig. 2(c) shifting more than one bit at each routing step clearly shortens the lookup latency. One problem with Koorde is that it is hard to setup the De Bruijn pointers in large networks if the network is bootstrapped very fast. If too many nodes do not have a valid De Bruijn pointer and fall back to their successor lists for routing, De Bruijn stabilize requests get lost due to exceeded hop count limits.

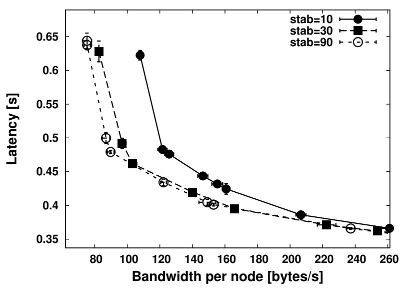
TABLE II: Parameter values for Koorde

Parameter	Value
Size of successor list	8, **16**, 32
Stabilization interval	**10 s**, 30 s, 60 s
De Bruijn interval	30 s, **120 s**, 480 s
Bits per digit b	1, 2, **4**, 6
Check predecessor interval	**10 s**, 30 s, 120 s

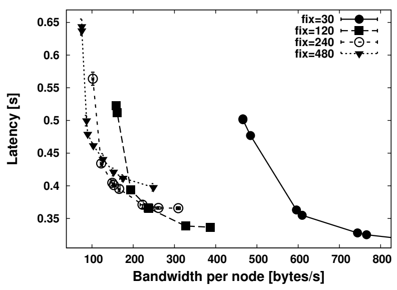
3) Pastry: Fig. 3 shows Pastry's results. In Fig. 3(a) Pastry in its original version is compared with the second proposal using periodic tasks and no second stage. The original version is evaluated with different values for the *Neighborhood Set* size. The results attest the advantages of the newer proposal in latency as well as in bandwidth consumption. The original version achieves lower latencies only with unacceptable high bandwidth consumption due to its expensive joining procedure. A bigger neighborhood set just increases the amount of traffic.

Although Pastry was proposed with a value of 4 bits per digit, Fig. 3(b) shows that when using a value of 2 or 4, no lower latencies can be achieved unless with a huge amount of additional traffic. With higher values for b, nodes are probed less often leading to higher latencies in churn scenarios due to failed nodes in the routing table. *Leaf Set* sizes of 4, 8, 16, and 32 were fixed in Fig. 3(c). Due to the exchange of complete states between nodes in a Pastry network, this parameter has a significant influence on bandwidth consumption while latencies only differ marginally. Though not proposed, Pastry was simulated with iterative routing mode as well (not illustrated here), the results show significant higher latencies in iterative mode compared to recursive mode. As no redundancy was provided in *find node response* messages, iterative lookups often aborted due to node failure.

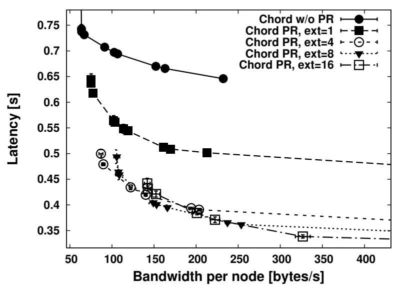
4) Bamboo: Fig. 4 illustrates Bamboo's results. Fig. 4(a) shows the convex hulls for different numbers of bits per digit. Contrary to Pastry, a value of 2 leads to very small latencies without a significant higher bandwidth consumption. As in Pastry, higher values just lead to a higher traffic amount. Like



(a) Chord stabilize interval (s)

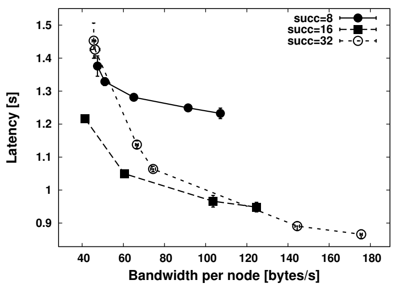


(b) Chord fix fingers interval (s)

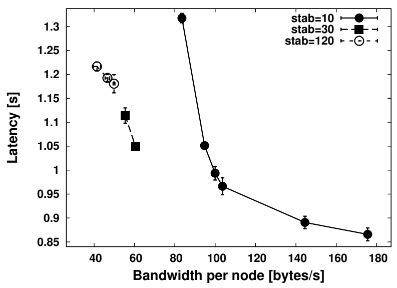


(c) Chord proximity routing

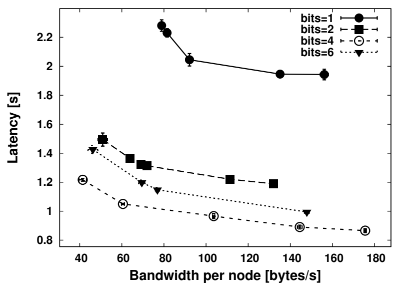
Fig. 1: Chord with parameter convex hulls



(a) Koorde De Bruijn and succ. size

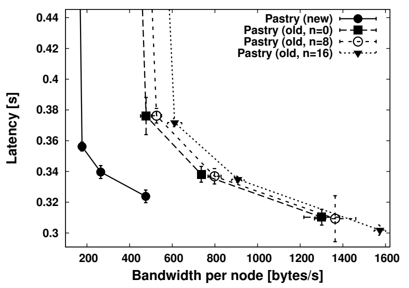


(b) Koorde stabilize interval (s)

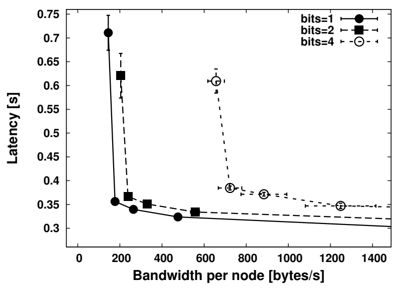


(c) Koorde shifting bits

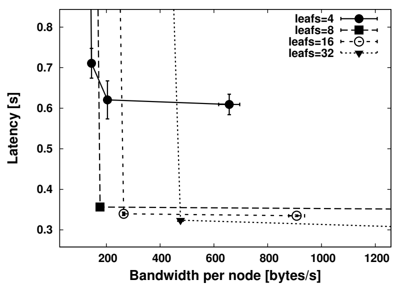
Fig. 2: Koorde with parameter convex hulls



(a) Pastry old vs. new / neighbors



(b) Pastry bits per digit



(c) Pastry leaf set size

Fig. 3: Pastry with parameter convex hulls

TABLE III: Parameter values for Pastry

Parameter	Value
Leaf Set size	4, **8**, 16, 32
Neighborhood Set size / Version	**0 (new)**, 0 (old), 8 (old),16 (old)
Bits per digit *b*	**1**, 2, 4

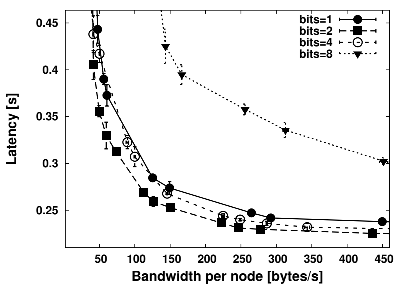
Pastry, Bamboo does not need a high number of leafs in its leaf set under typical churn. Our results show that 8 leafs are enough to keep the network in a stable state (figure omitted).

In Fig. 4(b) the leaf set maintenance task interval was altered. As shown, an interval of 60 seconds is fair enough to keep the leaf set consistent.

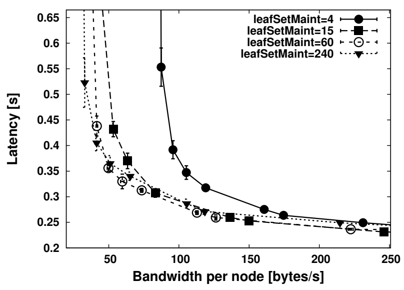
The *Local Tuning* interval has marginal effect on lookup latencies (figure omitted). Best results are achieved with intervals of 4 or 10 seconds. If it is set to 150 seconds, higher latencies must be faced. The significant impact of different *Global Tuning* intervals is shown in Fig. 4(c). As expected, this also periodically triggered mechanism decreases latencies without high costs. A value of 60 seconds is adequate, if lower latencies are requested and bandwidth consumption of more than 100 Bytes/s are acceptable, a value of 10 should be chosen.

TABLE IV: Parameter values for Bamboo

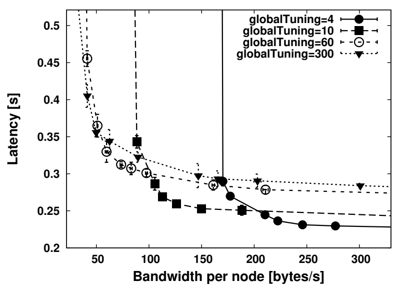
Parameter	Value
Leaf Set size	4, **8**, 16, 32
Leaf Set maintenance interval	4 s, 15 s, **60 s**, 240 s
Local Tuning interval	4 s, **10 s**, 30 s, 150 s
Global Tuning interval	**10 s**, 60 s, 300 s, 600 s
Bits per digit *b*	1, **2**, 4, 8



(a) Bamboo bits per digit

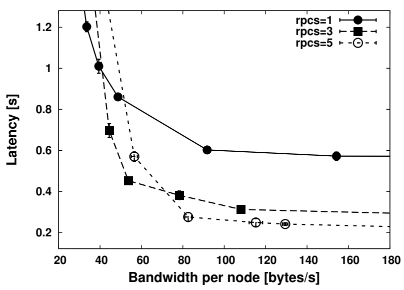


(b) Bamboo leaf set maint. interval (s)

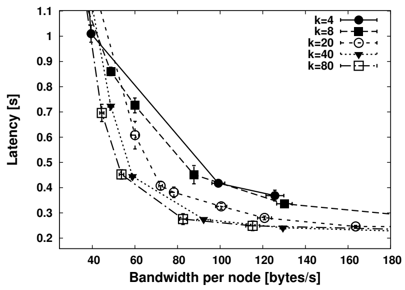


(c) Bamboo global tuning interval (s)

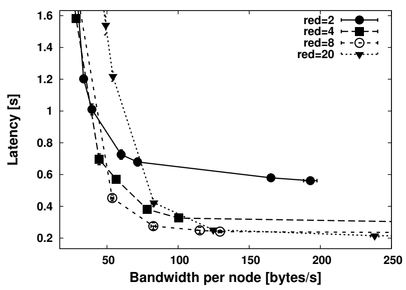
Fig. 4: Bamboo with parameter convex hulls



(a) Kademlia parallel RPCs



(b) Kademlia k



(c) Kademlia number of red. nodes

Fig. 5: Kademlia with parameter convex hulls

5) Kademlia: The results of Kademlia's simulations are shown in Fig. 5. Our evaluation showed that the parameter b (*bits per digit* should have a value of 1 to get the best performance as well as the lowest bandwidth consumption (figure omitted). Increasing the value of k (shown later) is always better than a high value for b. In Fig. 5(a) different numbers of parallel RPCs were simulated. The lookup latencies can obviously be decreased by sending more than one RPC at the same time, as only one node must be alive and respond to continue the lookup. Additionally, responses from physically closer nodes are received earlier which accelerates the lookup procedure. As a matter of course, using parallel RPCs comes with an increased amount of traffic.

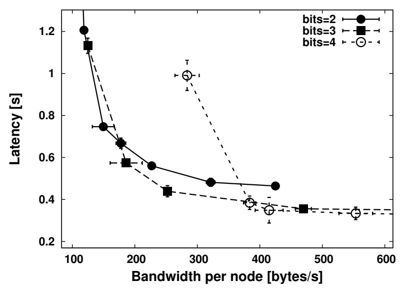
The bucket size k was altered for Fig. 5(b). A considerable difference between a value of 4, 8 and 20 is noticeable. With values of 40 or 80, better results can be achieved concerning lookup latencies. As there is no periodically triggered stabilization protocol, larger routing tables do not lead to more stabilization traffic. The impact of altering the numbers of redundant nodes—the nodes which are returned in a find node response message—is shown in Fig. 5(c). Here, best results are achieved with a value of 8. A value of 20 redundant nodes only lead to lower latencies using a minimum of 200 Bytes/s. Combined with the results shown in Fig. 5(b), using a value $r < k$ is apparently reasonable to achieve low latencies while keeping the lookup traffic small.

The results for the bucket refresh interval are presented without Figure. As the test application used for the simulations periodically sends probe messages to random destination keys, most buckets are refreshed by application triggered lookups. When using an interval of 4,000 seconds, no stabilization lookups are performed at all due to a sufficient number of application triggered lookups. An interval of 1,000 seconds seems to be a good compromise in this simulation scenario to keep the buckets up-to-date. Shorter intervals do not have any positive effect in this particular scenario, rather they come with higher costs.

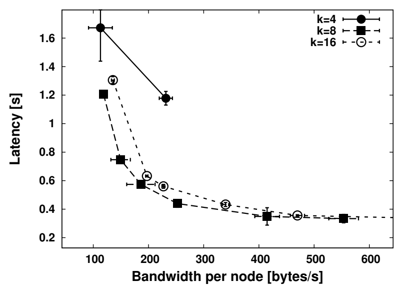
TABLE V: Parameter values for Kademlia

Parameter	Value
Bucket size k	4, 8, 20, **40**, 80
Number of returned nodes per hop r	2, 4, **8**, 20
Bits per digit b	**1**, 2, 4
Bucket refresh interval	100 s, **1000 s**, 4000 s
Number of parallel lookups α	1, 3, **5**

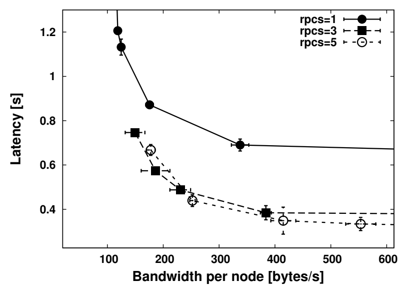
6) Broose: Broose has a very expensive join process since a joining node queries the nodes at the joining location for the complete routing table contents. Since the number of bits per digit directly influences the routing table size, a large number of bits per digits leads to even higher communication for the join process (see Fig. 6(a)). Unlike in Kademlia routing tables in Broose are very restricted. Therefore a large bucket is mainly needed for redundancy (like the successor list in Chord) and can not be used to reduce the number of routing steps. As shown in Fig. 6(b) a bucket size larger than $k = k' = 8$ does not improve lookup performance in the evaluated churn scenario. Due to the exhaustive join process Broose can not benefit from its constant node degree and needs even more bandwidth than Kademlia with $O(logN)$ buckets.



(a) Broose shifting bits



(b) Broose bucket size



(c) Broose parallel RPCs

Fig. 6: Broose with parameter convex hulls

Like with Kademlia the number of parallel RPCs plays a major role on lookup performance. As shown in Fig. 6(c) increasing the number of parallel RPCs to 5 significantly decreases lookup latencies with only a very moderate increase in bandwidth.

TABLE VI: Parameter values for Broose

Parameter	Value
Bucket size k and k'	4, **8**, 16
Bits per digit b	2, **3**, 4
Bucket refresh interval	30 s, **60 s**, 300 s
Number of parallel lookups α	1, 3, **5**

B. Routing variants

In Fig. 8(a), all the routing modes that are applicable for Chord were compared. No proximity routing is applied. As expected, the semi-recursive routing mode achieves lowest latencies by far. All recursive modes show better results than the iterative alternative without redundancy. Due to the fact that full-recursive routing in Chord means to go around the whole ring always, source-routing-recursive mode is slightly faster. The impact of activated redundancy in iterative mode is noteworthy, as similar results in lookup latencies compared to source-routing-recursive and full-recursive mode are achieved. Fig. 8(b) shows the effect of acknowledgement in recursive routing mode using Chord (without PR). Apparently, acknowledgements are essential for recursive routing in churn scenarios and do not come with increased bandwidth consumption. The difference between iterative and exhaustive-iterative lookups is shown in Fig. 8(c) using the Kademlia protocol, as only Kademlia supports both modes. As expected, the exhaustive mode —originally proposed in [17]— is much more expensive concerning bandwidth consumption. As all potential siblings have to be contacted, lookup latencies in exhaustive mode are about 10 times higher.

C. Protocol comparison

Fig. 7 shows all simulated protocols with parameters optimized for this network scenario. While Bamboo and Kademlia show the best performance/cost tradeoff, Chord can not keep up even in recursive routing mode using proximity routing. Iterative Chord suffers from very high latencies, while Broose shows a lack of stability. For efficient routing Broose needs a precise estimation of the network size, which can only

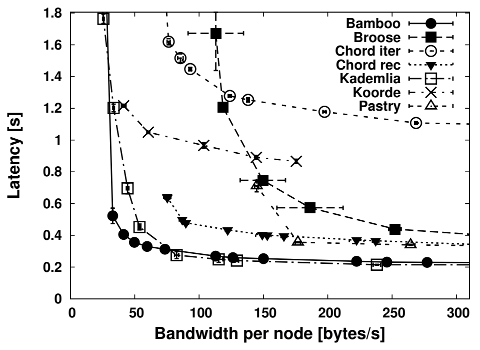


Fig. 7: Comparison of KBR protocols with optimal parameters

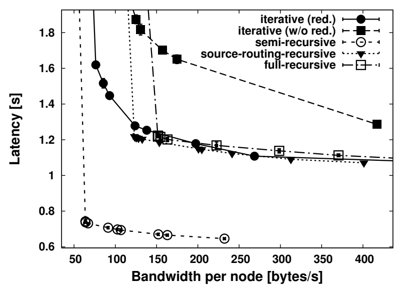
be achieved by sending additional messages. Koorde cannot compete at all as a result of its longer routing paths. This results in a poor performance/cost tradeoff for both De Bruijn based protocols. Pastry only achieves low latencies combined with high bandwidth consumption.

D. Churn

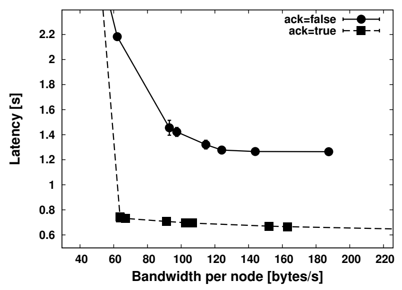
In this section, the best parameter configurations identified in the simulations before were fixed, while the nodes' lifetime mean was altered. Fig. 9(a) shows the KBR delivery ratio of all protocols under different churn rates. Pastry achieves success rates of about 85%, even under extreme churn (100 s lifetime mean), while Bamboo, and Kademlia come with success ratios of 60%. Due to its expensive joining procedure, Pastry's routing tables stay up-to-date in high churn scenarios. Chord shows stability from 1,000 seconds lifetime mean upwards. Koorde achieves only 75% in scenarios with 10,000 seconds lifetime mean of nodes.

In Fig. 9(b) the bandwidth consumption of all protocols is shown under different churn rates. Here, Pastry produces immense traffic under nearly all churn rates due to its expensive joining procedure. Kademlia has an even higher traffic amount as Pastry, but only under extreme churn. Bamboo shows constant small traffic as a result of its periodic tasks and minimal node arrival operation. Chord shows an average bandwidth consumption, but does not stabilize under high churn. Koorde also collapses under high churn, so the bandwidth results have no significance.

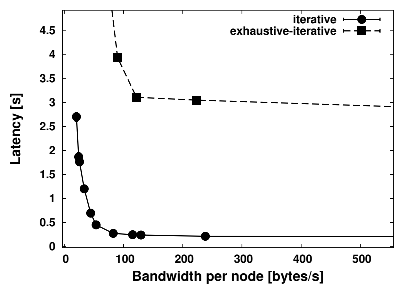
The measured lookup latencies of successfully delivered



(a) Recursive routing (Chord)

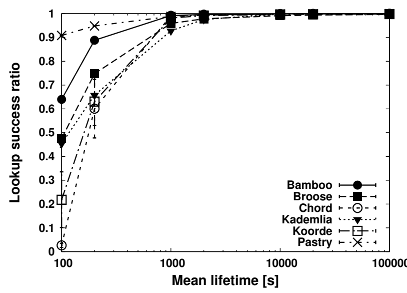


(b) Acknowledgements (Chord)

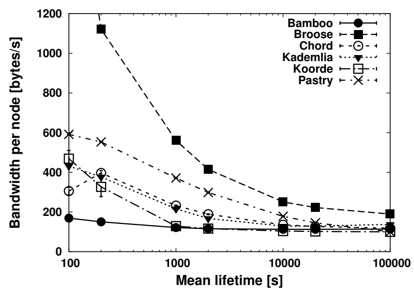


(c) Iterative vs. exhaustive-iterative (Kademlia)

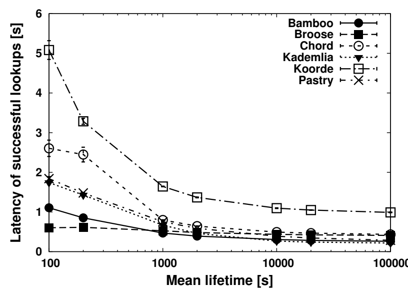
Fig. 8: Routing modes



(a) Delivery ratio under churn



(b) Bandwidth under churn



(c) Latency under churn

Fig. 9: Comparison of KBR protocols under different churn settings with 10,000 nodes

messages are plotted in Fig. 9(c). Kademlia and Bamboo both achieve very small latencies—even under churn—due to the use of *proximity neighbor selection (PNS)* or parallel RPCs, respectively. While Pastry and Chord show acceptable results, Koorde collapses under high churn rates, which was previously shown in Fig. 9(a): Nodes have no up-to-date De Bruijn list, thus routing must be done inefficiently using the successor list.

E. Scalability

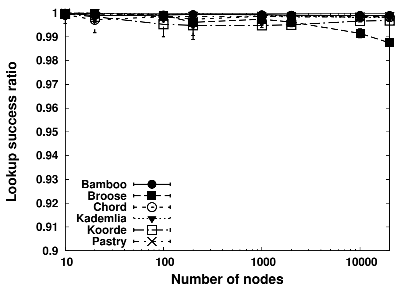
In this section all protocols are tested for scalability in scenarios with 10 - 20,000 nodes. Like in the last section, the best parameter configurations identified in the simulations before were fixed, while the total number of nodes was altered. Fig. 10(a) shows the results for KBR delivery ratio. All protocols deliver messages reliably, regardless of the network size.

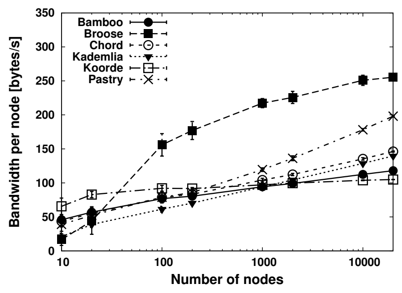
In Fig. 10(b) the bandwidth consumption is shown. All protocols show a nearly logarithmic increase of network traffic per node. Bamboo shows the best scalability, while Pastry cannot keep up due to its strongly increasing traffic amount. Kademlia and Chord show acceptable results. Fig. 10(c) shows logarithmic increase of lookup latency for all protocols except for Koorde. Broose and Kademlia show the lowest latencies in small networks due to their large routing tables. Both protocols have a higher increase of latencies with larger numbers of nodes, though.

F. Summary of Results

The most remarkable results of our simulations can be summed up as follows:

- *Bamboo* and *Kademlia* show the best results regarding lookup latencies and bandwidth consumption. Bamboo benefits from proximity neighbor selection, acknowledgements and periodic maintenance. Kademlia achieves its best results with our proposed simple iterative routing mode using several parallel RPCs instead of using the exhaustive-iterative mode.
- For *Kademlia* an increased bucket size k in combination with a small number r of returned nodes leads to lower latencies while keeping the maintenance traffic small.
- *Chord* and especially *Pastry* show only average results and they do not have any benefits over *Bamboo* – still they are widely proposed as basis for new P2P protocols (e.g. the new *RELOAD Internet draft* [21] for distributed VoIP is based on *Chord*).
- A high number of parallel sent RPCs in iterative lookups significantly decreases lookup latencies. As closer nodes (in terms of proximity) send their responses earlier, this can be considered as kind of *proximity routing* for iterative lookups. In this way iterative *Kademlia* can catch up with recursive *Bamboo* in medium sized networks. Although security aspects were not in our focus, it should be noted that *Kademlia*'s iterative lookups might have security benefits compared to recursive routing.
- Protocols based on De Bruijn graphs like *Koorde* and *Broose* cannot keep up with the other protocols due to problems to maintain the topology under churn.
- In recursive routing mode, the usage of acknowledgement messages has high impact on lookup latencies. Further-


(a) Delivery ratio vs. network size


(b) Bandwidth vs. network size

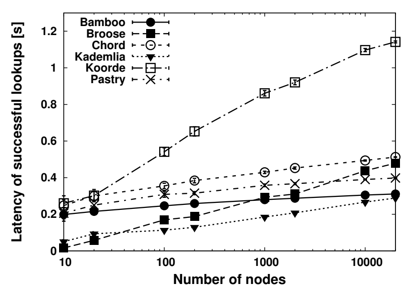

(c) Latency vs. network size

Fig. 10: Comparison of KBR protocols under different network sizes with 10,000s mean lifetime

more, the importance of periodically triggered routing table maintenance processes decreases, as failed nodes are mostly detected during lookup procedures.

VI. Conclusion

In this paper we compared six state of the art KBR protocols using our simulation framework OverSim. Many protocol parameters were analyzed in terms of influence on performance and bandwidth costs. We compared several iterative and recursive routing modes and the usage of acknowledgements for KBR routing. Finally, all protocols were evaluated in different churn scenarios and tested for scalability.

Overall, Kademlia and Bamboo show the best performance vs. cost tradeoff, where Kademlia highly benefits from the usage of faster iterative lookups compared to the original proposal. Both De Bruijn based protocols show a lack of stability, in particular in high churn and large-scale scenarios.

We hope that the results presented in this paper and our open-source simulation framework OverSim will help researchers and designers of P2P systems to choose the right protocol and to tune its parameters according to their requirements and network characteristics.

References

[1] I. Baumgart, B. Heep, and S. Krause, "OverSim: A Flexible Overlay Network Simulation Framework," in *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007*, Anchorage, AK, USA, May 6–12, 2007, pp. 79–84.

[2] I. Baumgart, B. Heep, C. Hübsch, and A. Brocco, "OverArch: A common architecture for structured and unstructured overlay networks," in *Proceedings of the* 15^{th} *IEEE Global Internet Symposium in conjunction with IEEE INFOCOM 2012*, Orlando, FL, USA, Mar. 2012.

[3] D. Loguinov, J. Casas, and X. Wang, "Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience," *IEEE/ACM Trans. Netw.*, vol. 13, no. 5, pp. 1107–1120, Oct. 2005.

[4] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The Impact of DHT Routing Geometry on Resilience and Proximity," in *Proceedings of the ACM SIGCOMM '03 conference*, Karlsruhe, Germany, 2003, pp. 381–394.

[5] D. Stutzbach and R. Rejaie, "Improving Lookup Performance Over a Widely-Deployed DHT," in *Proc. 25th IEEE International Conference on Computer Communications INFOCOM 2006*, Barcelona, Catalunya, Spain, Apr. 2006, pp. 1–12.

[6] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI '04)*, San Francisco, California, USA, Mar. 2004.

[7] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Communications Survey and Tutorial*, vol. 7, pp. 72–93, 2005.

[8] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT," in *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, Boston, MA, USA, Jun./Jul. 27–2, 2004.

[9] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil, "A performance vs. cost framework for evaluating DHT design tradeoffs under churn," in *24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, vol. 1, Miami, FL, USA, Mar. 13–17, 2005, pp. 225–236.

[10] N. I. Damm, D. Fahrenholtz, and V. Turau, "On Fluctuation Resilience of Second Generation Distributed Hash Tables," in *Kommunikation in Verteilten Systemen (KiVS 2007)*, T. Braun, G. Carle, and B. Stiller, Eds., 2007, pp. 105–110.

[11] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, Feb. 2003.

[12] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Exploiting network proximity in distributed hash tables," in *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, Bertinoro, Italy, Jun. 3–7, 2002, pp. 52–55.

[13] M. F. Kaashoek and D. R. Karger, "Koorde: A Simple Degree-Optimal Distributed Hash Table," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, vol. 2735/2003, Berkeley, CA, USA, 2003, pp. 98–107.

[14] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Middleware 2001 : Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, vol. 2218/2001, Heidelberg, Germany, Nov. 12–16, 2001, pp. 329–350.

[15] M. Castro, P. Druschel, and Y. C. Hu and Antony Rowstron, "Topology-Aware Routing in Structured Peer-to-Peer Overlay Networks," Microsoft Research, Redmond, WA, Tech. Rep. MSR-TR-2002-82, 2002.

[16] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT," EECS Department, University of California, Berkeley, CA, USA, Tech. Rep. UCB/CSD-03-1299, Dec. 2003.

[17] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Peer-to-Peer Systems: First International Workshop (IPTPS 2002). Revised Papers*, vol. 2429/2002, Cambridge, MA, USA, Mar. 7–8, 2002, pp. 53–65.

[18] A.-T. Gai and L. Viennot, "Broose: a Practical Distributed Hashtable based on the De-Bruijn Topology," in *Fourth International Conference on Peer-to-Peer Computing (P2P 2004)*, Zurich, Switzerland, Aug. 2004.

[19] D. Stutzbach and R. Rejaie, "Understanding Churn in Peer-to-Peer Networks," in *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, Rio de Janeiro, Brazil, Oct. 25–27, 2006, pp. 189–202.

[20] M. Steiner, T. En-Najjary, and E. W. Biersack, "Long Term Study of Peer Behavior in the KAD DHT," *IEEE/ACM Transactions on Networking*, vol. 17, no. 6, pp. 1371–1384, Oct. 2009.

[21] C. Jennings, B. B. Lowekamp, E. Rescorla, S. A. Baset, and H. Schulzrinne, "Resource location and discovery (reload)," IETF Internet-Draft, work in progress, draft-ietf-p2psip-base-18, Aug. 2011.