Christian Groß, Björn Richerzhagen, Max Lehn:
Benchmarking Peer-to-Peer Systems. In: Wolfgang Effelsberg, Ralf Steinmetz, Thorsten Strufe, vol. 7847, vol. LNCS, chap. Structured Search Overlays, p. 49-67, Springer, June 2013. ISBN 978-3-642-38672-5.
http://www.springer.com/computer/communication+networks/book/978-3-642-38672-5.

Chapter 4
Structured Search Overlays

Christian Groß, Björn Richerzhagen, Max Lehn

Since the first structured search overlays for peer-to-peer systems have been presented in the years 2001/2002, a variety of approaches have been developed, greatly differing in their design. Although all of these approaches provide almost the same functionality, their respective evaluations greatly differ with respect to the metrics and workloads used. These differences in the evaluations make it hard, if not impossible to compare their performance and resulting costs. In addition, most evaluation sections do not provide information about the performance limits of the respective structured search overlays.

To cope with the problem of comparability, Li et al. [7] presented a performance vs. cost framework for evaluating DHTs under churn. Although the approach can be used to compare different overlays with respect to their performance and cost under churn, Li's approach only evaluates the performance and costs of DHTs under churn and does not take into account other environmental conditions such as an increasing message loss or an increased service consumption resulting in a higher workload for the DHT. In contrast, the presented benchmarking approach investigates the performance and costs in a holistic fashion with the specific dedication in determining the performance limits of a structured search overlay.

Therefore, a basic benchmark for structured search overlays along the formal model and the methodology established in Chapters 2 and 3 is defined. The benchmark addresses two main goals: First, it allows to compare existing structured search overlay implementations under different workloads. Based on this comparison, it is

Christian Groß
Technische Universität Darmstadt, Multimedia Communications Lab, Darmstadt, Germany,
e-mail: chrgross@kom.tu-darmstadt.de

Björn Richerzhagen
Technische Universität Darmstadt, Multimedia Communications Lab, Darmstadt, Germany,
e-mail: richerzhagen@kom.tu-darmstadt.de

Max Lehn
Technische Universtität Darmstadt, Databases and Distributed Systems Group, Darmstadt, Germany, e-mail: max_lehn@dvs.tu-darmstadt.de

possible to determine which overlays are suitable for a specific application scenario, stating specific workload characteristics. Second, by pushing structured search overlay implementations to their performance limits, their strengths and weaknesses become visible.

Before coming up with a concrete interface definition for the benchmark, it is important to first define structured search overlays and their functionality. Throughout this chapter, we define structured search overlays along the lines of the definition given in [11], as an overlay network that enforces a predefined topology for interconnecting peers. The overlay protocol ensures that the routing process is deterministic by using a particular data structure for storing overlay contacts, usually referred to as the routing table.

The fundamental functional block that each structured search overlay provides is to route a given message m with hash key k from a peer p to a responsible peer q. The hash key k usually is computed by using existing hash functions such as MD5 or SHA-1. On top of this basic routing function, a search functionality for data objects or peers can be built. This is done by hashing data objects or peer information to determine their corresponding hash key k. Based on the key, the overlay routes search requests to the position in the overlay where either the peer or the data object of interest is located.

4.1 Interface Definition

When designing a benchmark for structured search overlays, one has to think about which functional block the benchmark should address. If the routing functionality is to be tested, the interface definition must contain a single `route(`k`, `m`) → flag` method. This method takes the hash key and message as an input and asynchronously returns a boolean value (the flag) indicating whether the message was routed successfully to the responsible peer or not. Several approaches exist for generating particular hash keys that are requested via the interface, as discussed in Section 4.3.2.2.

When the search functionality is to be benchmarked, the interface definition must be extended such that the storage and retrieval of data objects is supported. Therefore, the two methods `put(`k`, data, lifetime)` and `get(`k`) → data` for storing and retrieving an object under a given key k are added to the interface. In addition, we assume a `getDirectNeighborSet() → neighbors` method, which returns the set of neighboring peers of a peer in the overlay. The set of neighbors is defined as the list of the closest peers to a peer p measured by the same distance metric as used by the route function.

The following assumptions are made: (i) The `get` function asynchronously returns the result, meaning that the result for a request is returned later on through a callback function. (ii) It is expected to always deliver a result. (iii) If no item for the requested key is found, the overlay returns an empty object. (iv) Each object is stored with a maximum lifetime after which it is deleted. Thereby, a simple garbage

collection mechanism is realized, avoiding peers to store outdated information and to get overloaded.

4.2 Non-Functional Requirements

When looking at the non-functional requirements, a search overlay should *scale* with respect to the number of users participating, the number of objects being stored, and the number of request generated by users in the overlay.

In addition, it should be *stable* even under high churn rates, meaning that the success ratio and the average query response time should not drop below a given threshold. For most applications that use a search overlay, a success ratio and recall close to one and a response time below one second is acceptable. Furthermore, the search overlay should be *robust* against a massive leave or join of peers. From the *fairness* point of view, the overlay should distribute the load according to the capacities of the peers while providing equal access to the resources of the overlay to all peers. In doing so, overloaded or starving peers in the overlay are avoided. The overlay should always deliver a *valid* answer to a request, i.e., the *correct* object for a given key is to be returned, or an empty object if no correct one can be found.

4.3 Workload

The generation of a workload for structured search overlays is based on their functional interface. As discussed in Section 4.1, we assume two types of interfaces and, thus, two types of workloads. To test the routing functionality, a *synthetic* peer lookup workload model is defined that generates peer lookups for online peers in the overlay. Defining an *application workload* for testing the routing functionality is not possible as there is no peer-to-peer application that uses only the routing functionality of structured search overlays.

We use two different models to generate the load for the overlay: (1) a synthetic workload model where requests and updates for abstract objects are generated artificially, with the goal of determining the performance limits of existing structured search overlay implementations, (2) an application-driven workload model representing a distributed overlay-based BitTorrent tracker, where peers request information about which other peers are currently participating in a particular swarm.

4.3.1 Synthetic Peer Lookup Workload Model

To model peer lookups we define a peer oracle that stores the information about which peers are currently online in the overlay. In this way, we are able to generate

requests for online peers only. Workloads are comprised of two types of parameters: (1) parameters for modeling the geographical distribution of peers and their online behavior and (2) parameters for modeling the route requests generated by the peers. The workload model has the following set of parameters:

Number of Peers, Session and Intersession Times, First Join, Final Leave. One of the basic parameters is the total number of peers that participate in the overlay. For each peer in the overlay, we model its lifetime in accordance with Chapter 3, which comprises the following phases: (i) initial join, (ii) presence phase, (iii) multiple repetitions of leave and join phases, (iv) final leave, meaning that the peer will not rejoin the system any more for the rest of its lifetime. To model this behavior, the following parameters are required: (i) the session time t_s, (ii) the time between two sessions t_i, (iii) the probability $P(\text{leave})$ for a leaving peer to not return to the system, and (iv) in case of a peer joining the system the probability $P(\text{new})$ that the peer joins the system for the first time. The impact of selecting entirely new peers for joining the system is that new peers do not have any knowledge of earlier sessions, e.g., stored objects.

Geographical Distribution of Peers. Another important parameter is the geographical distribution of peers, which has an influence on the delay between peers and the resulting response time of the structured search overlay under test.

Peer Activity. Finally, peers generate requests. The behavior of the peers is modeled as a Poisson process with arrival rate or intensity λ_r. In a Poisson process with intensity λ the inter-arrival times between consecutive events are exponentially distributed with arrival rate λ. In addition, the popularity distribution of route requests to target peers is needed. This popularity distribution can be modeled using measurements of real applications or synthetic popularity distributions.

4.3.1.1 Per Peer Workload Generation

To generate load on the system under test the following method is used: First, a peer draws a value for the inter-request timer that the peer has to wait between two successive route requests. The request behavior is modeled as a Poisson process, where the inter-request times follow a exponential distribution. Afterwards, the peer waits until the execution time for the next route request is reached. Then the peer draws a random peer ID, which is used as the target ID for the route request, and calls the route method and passes the drawn ID to it. Finally, the peer waits until either a result for his request is returned or the timeout expires.

4.3.1.2 Workload Scenario

In the following, the peer variation schemes for the peer lookup workload are presented. The workload scenarios are derived from the generic workload scenarios presented in Section 3.4.1. They are grouped according to the elementary entities of

the structured search overlay. These are (i) the peers, (ii) the services provided by the peers, and (iii) the underlay.

Peer Parameterization

- *Scenario 1: Without Churn.* Peers join the network, and after a static period, where no further join or leave of peers occurs, the workload is deployed on the system. This workload scenario should demonstrate the performance and costs of the structured search overlay under ideal conditions, without peer churn. If a SUT is not capable of providing a reasonable performance at reasonable costs in this scenario, it suffers from severe design drawbacks.
- *Scenario 2: Exponential Churn.* In Scenario 2 the performance of the structured search overlay is investigated with an increasing level of peer churn. We execute multiple runs during which the churn factor of the exponential model is decreased stepwise. Peers initially join the system similar to Scenario 1, and the system stabilizes in a silent period. Afterwards, the churn is enabled as follows: The exponential churn model defines the session time λ_s and intersession time λ_i which both are decreased from λ over $\frac{1}{2}\lambda$, $\frac{1}{4}\lambda$, $\frac{1}{8}\lambda$ to $\frac{1}{16}\lambda$.
- *Scenario 3: Massive Crash.* The third scenario covers the extreme situation of a large fraction of peers crashing. As in the two scenarios before, peers join, and the workload starts after a silent period. The percentage of peers leaving the overlay ungracefully is increased stepwise per run from 10% to 90%.
- *Scenario 4: Massive Join* In contrast to Scenario 3, the massive join scenario deploys a simultaneous join process of a large number of peers instead of a massive crash. Initially, peers join the overlay, and after a static period, a massive join takes place. The percentage of peers suddenly joining the overlay is increased stepwise per run from 10% to 90%.
- *Scenario 5: Increasing Number of Peers.* Peers join the network according to a linear function, increasing their number as long as the system remains stable.

Service Parameterization

- *Scenario 6: Flash Crowd.* In this scenario, a large number of the peers request a specific content in a short amount of time. Again, a join and silent phase is assumed, like in the last settings. Peers join the overlay, and the exponential churn model is deployed together with the aforementioned workload model. Route requests per peer are modeled as a Poisson process with an arrival rate of λ_r request per minute. After a silent period, the flash crowd begins. The intensity of the flash crowd is doubled per run, meaning that the average number of requests executed per peer is doubled per run until it reaches 16 times the initial load level.
- *Scenario 7: Increasing Service Consumption.* We deploy the structured search overlay under test like in the scenarios before and increase the intensity λ_r of the

Poisson process that models the request frequency of peers. Over multiple runs, the intensity λ_r is doubled per run until in reaches 16 times the initial value.

Network Parameterization

- *Scenario 8: Increasing Message Loss.* In this scenario the delivery reliability is decreased by increasing the percentages of messages being dropped. Similar to Scenario 2, peers join the overlay and churn is enabled. After a silent period, the rate of messages being dropped by the underlay is increased stepwise per run from 1% to 2%, 5%, and 10%.

4.3.2 Synthetic Object Lookup Workload Model

In contrast to the peer lookup workload where peers execute route requests, peers in the data lookup workload execute store and search requests for objects. Objects are modeled with a given maximum lifetime, which are stored in the overlay and retrieved afterwards. All objects that are stored in the overlay are also stored in a *Global Object Database* which is not part of the SUT. Since this benchmark is designed for simulated or emulated environments, we assume that this database can be maintained as part of the global knowledge in the simulator. The database is used for selecting objects to be queried as well as for validating results obtained from the overlay. The following parameters are used in our model:

Number of Peers, Online Time, Non-persistent Storage on Peers. An important parameter is the number of peers, as it directly determines the load for the overlay. The peers' online times are determined by the churn model, which describes the session and inter-session times. In our workload model we assume a non-persistent storage in case that a peer goes offline, which means that its stored data is deleted. In doing so, we avoid the re-insertion of possibly outdated objects into the overlay caused by rejoining peers.

Object Size, Popularity, Lifetime. The second parameter set is related to the objects that are to be stored in the overlay. We model the popularity of objects according to a Zipf distribution [9], [6] . Stored objects are modeled with a fixed size. In order to avoid a constantly growing number of objects, we introduce an object lifetime, after which an article is considered to be outdated and i sdeleted from the overlay.

Peer Activity. The third set is related to the peer activity, specifying how often a peer executes a certain type of action. We define three basic operations: creating a new object, requesting an existing object, and updating an object. Hence, it is necessary to specify an execution probability per peer for each of these operations. In addition, the time between successive operations is needed. A grace period after the creation or update of an object before a read or update request for the same object allows the overlay to properly store the objects. The delete

operation is not part of the peer activity. Objects are deleted automatically by the overlay as soon as their lifetime expires.

As already mentioned, the *Global Object Database* maintains information about all objects stored in the overlay. For each object this information comprises the object id, the object lifetime, the object store timestamp, and a hash value of the object. This information is needed in order to verify whether the correct version of a requested object is returned by the overlay. The object database offers methods for creating, updating, and requesting an object.

As mentioned above, in order to obtain meaningful results in the benchmark for the structured search overlays, the overlays have to provide means for replicating objects in order to prevent the loss of stored objects in the case of a leaving or failing peer.

4.3.2.1 Per Peer Workload Generation

The workload generation algorithm, which is run by each peer, works as follows. Initially, an activity index is defined per peer, drawn from a global activity index distribution. Similar to the peer lookup workload, the requests issued by a particular peer are modeled as a Poisson process with a given arrival rate λ_r. The activity index defines the expected value of inter-arrival times between two successive actions performed by the peer. In contrast to the workload model for the node lookup, peers can decide between three different functional methods: (i) storing a new data object, (ii) requesting a data object, or (iii) updating an already stored data object. The update method can be realized by first issuing an lookup for a data object, followed by a store request. To model the decision process, a peer draws a uniform distributed random value between 0 and 1. For each method a certain probability range is defined, with all the ranges summing up to 1. Based on the drawn probability value, the method is select by calculating the range that the drawn probability value is overlapping with. Finally, the chosen method is executed and a new inter-operation time is drawn, which determines how long a certain peer has to wait until it executes the next operation.

4.3.2.2 Workload Scenarios

For the workload, the scenarios as presented in Section 4.3.1.2 are assumed with some small differences. Instead of invoking the peer lookup algorithm, the peers execute the workload algorithm as described above.

4.3.3 Application-Based Workload Model

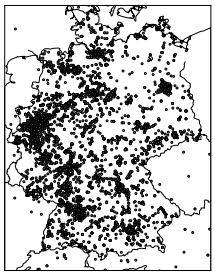
As already mentioned at the beginning of the Section, an application-based workload model representing a fully distributed BitTorrent tracker is proposed, where peers request information about which other peers are currently participating in a given BitTorrent swarm. In contrast to the synthetic workload, which tries to drive a system to its limits, the application-based workload aims at the generation of a realistic synthetic workload. Systems being benchmarked using an application-based workload can be compared to others in order to find the best system for a particular workload. The BitTorrent workload was derived from two large measurement studies presented in [4]. The first study investigated BitTorrent by periodically probing over 46,000 torrents in order to quantify high-level characteristics, such as the swarm size and the proportion of leechers and seeders. In contrast, the second study investigated relevant properties of BitTorrent users such as their download rates and chunk availability on a microscopic level by contacting over 700,000 individual peers in 832 torrents.

Furthermore, we model the geographical location of peers by using the distribution shown in Figure 4.1a, which was extracted from traces of Twitter [3], containing 22 million location-based status updates from 220,000 users.

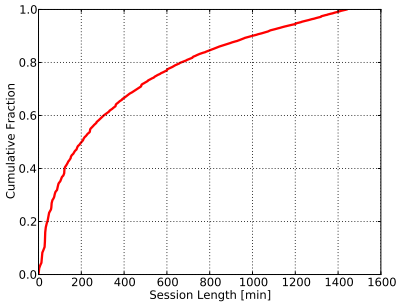
The typical workload for a distributed BitTorrent tracker works as follows: A peer joining a particular BitTorrent swarm performs a lookup in the overlay to get a list of peers that are currently participating in the swarm. The key used for the request is calculated by hashing the swarm's meta-data. A peer receiving the query request responds with a list of peers currently active in the swarm. Afterwards, the joining peer adds itself to the received list and stores the updated list in the overlay. To ensure the availability of all stored peer lists, they are replicated in the overlay.

In order to model the mentioned peer behavior, we extracted the workload parameters shown in Figure 4.1 from the measurements.

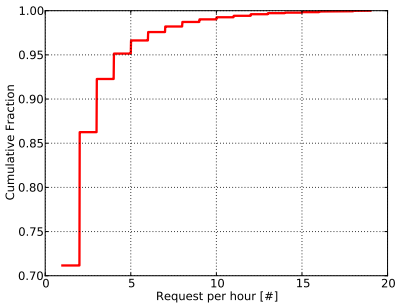
Firstly, each joining peer i selects the length of its session l_i, which determines how long the peer will remain in the system. This is done by drawing an equally distributed random number $r_l \sim U(0,1)$, which is mapped onto a session length based on the CDF shown in Figure 4.1b. Afterwards, a peer determines its session activity index λ_i from the CDF shown in Figure 4.1c by mapping a second random number $r_\lambda \sim U(0,1)$ onto the requests per hour. The activity index λ_i determines how many request the user has to perform per hour during its online time. The requests per peer are modeled as a Poisson process $P_{\lambda,t}$ with $\lambda = \lambda_i$ and $t \in [0,60]$ min. If the session length of a peer is smaller than a full hour the Poisson process is stopped at the end of the peer session. In case that the peer session length exceeds one hour, the Poisson process is repeated. For every request made by a peer the item to be request has to be determined. This is done by mapping a third random number $r_{item} \sim U(0,1)$ onto the item ID according to the CDF shown in Figure 4.1d.



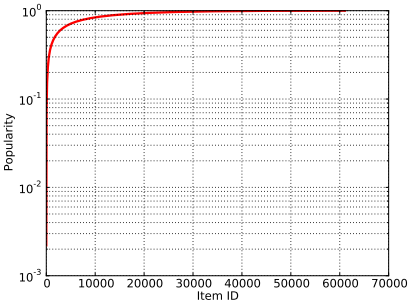
(a) Geographical Distribution of Peers



(b) CDF of the peer session length in minutes



(c) CDF of the peer request rates per hour



(d) CDF of item popularities

Fig. 4.1: BitTorrent workload parameters extracted form the measurements

4.4 Metrics

Having presented the synthetic as well as the application-based workload schemes, we are now going to present the metrics that are measured during the benchmark. Each set of metrics is associated with the corresponding quality aspect. First, we will introduce the basic metrics that quantify the performance and the cost of a system. Afterwards, we will present the derived metrics, that make use of the basic metrics. All metrics are measured on every peer i every Δt seconds, which results in a set of samples for each peer. Based on the set of samples global metrics can be calculated. The following basic metrics are measured per peer:

Performance The performance of a search overlay is quantified by two different metrics: The success ratio $s(i,t)$ and the query response time $t_q(i,t)$ at every peer i at time t.

Cost The costs for operating an overlay are quantified by the upload traffic $u(i,t)$ and download traffic $d(i,t)$ measured on each peer i at time t. The traffic can be further classified into maintenance and routing traffic.

For each metric x the Jain fairness index $F(x)$ is computed based on the averaged values per peer using the formula presented in Section 3.5.2. In doing so, the distribution of the performance and costs among the peers can be quantified. A fairness index close to one should be targeted as an unequal distribution of performance and costs either leads to peers leaving the system (unequal distribution of performance) or stability issues in case of an unequal distribution of costs.

4.5 Example Implementations

In the area of structured search overlays several attempts have been made that greatly differ in their design. A list of common characteristics of structured search overlays was derived in [1]. All approaches have in common that they maintain a fixed topology such as a ring or a tree. Peers and objects are mapped onto an identifier space using hash functions, e.g., SHA-1 or MD5. Based on the identifier space a distance metric is defined, which is used during the routing process for delivering search requests to their destination. Structured overlays use a greedy routing scheme, which ensures that the distance to the destination is decreased with each routing step. Prominent structured search overlays are Chord [14], Kademlia [8], and Pastry [10].

4.6 Benchmark Results

In the following the benchmark results for three structured search overlays are presented. All three overlays have been implemented in the discrete-event-based overlay network simulator PeerfactSim.KOM [13]. The benchmarking setup follows the methodology described above. All benchmarks are executed five times, and for all results the averages together with the 95th confidence intervals are reported. The values for the environmental setup, the workload, and for the concrete system parameters are shown in Table 4.1.

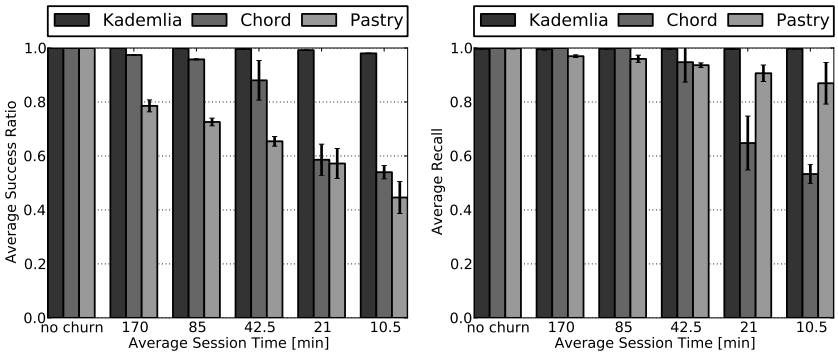
5,000 peers join the overlay and run the workload model described in Section 4.3.3. Delays are modeled according to the GNP delay model [5] as it provides a realistic model for approximating delays in the Internet. In scenarios with churn enabled, the KAD churn model [12] is used, which was derived from real measurements of the KAD overlay running together with BitTorrent. All benchmark runs are executed for twelve hours such that all operations of the overlay are executed multiple times.

Table 4.1: Environmental-, workload-, and system parameter setup.

Parameter	Value
Environmental and Workload Parameters	
Number of Peers	5,000
Workload Model	Application-based workload model based on BitTorrent measurements
Underlay Delay Model	GNP Delay Model [5]
Session Duration	KAD Churn Model [12] Weibull(λ_s, k_s), $\lambda_s = 169.5385$ min, $k_s = 0.61511$
Intersession Times	KAD Churn Model [12] Weibull(λ_i, k_i), $\lambda_i = 413.6765$ min, $k_i = 0.47648$
Simulation Duration	12 h
Chord System Parameters	
Size of Finger Table	160
Finger Table Update Interval	30s
Pastry System Parameters	
Size of ID Space	128 bit
Size of Leaf Set	10
Size of Neighborhood Set	10
Pastry Base Parameter b	4
Kademlia System Parameters	
Size of ID Space	160 bit
Number of Parallel Lookups α	3
Bucket Factor k	20

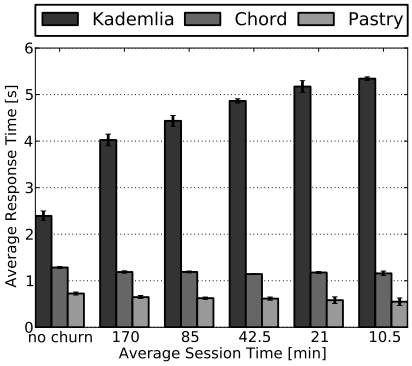
4.6.1 Stability

The results for the increasing churn benchmark are shown in Figure 4.2. Under ideal conditions all three overlays deliver a high success ratio and a recall of one, as shown in Figure 4.2a and 4.2b, indicating that all three overlay concepts work properly. When applying churn on all three overlays, the success ratio of Chord and Pastry is rapidly dropping whereas Kademlia shows a stable success ratio of one. The reason for the difference in the performance is that Kademlia uses an iterative routing scheme with parallel lookups, which is more robust to churn. In contrast to that, the recursive routing procedures used by Chord and Pastry are not able to cope with high churn rates. Especially Chord starts to deliver wrong results, as shown by the decreasing recall in Figure 4.2b. Furthermore, the iterative routing concepts with parallel lookups allows to reach a target peer via multiple paths, whereas the recursive ring routing of Chord and the prefix routing of Pastry are not that flexible. Here, requests are forwarded along a single path determined by the greedy routing procedure of both overlays. In case that a single peer along this path goes offline, the routing request is very likely to fail. This result is along the lines with the findings of a theoretical stability analysis presented in [2]. The stable performance of Kademlia under churn, however, comes at the cost of an increase response time, as shown in Figure 4.2c. The reason for this is again the iterative routing procedure, which needs one additional round trip time for sending back found peers within each step. For

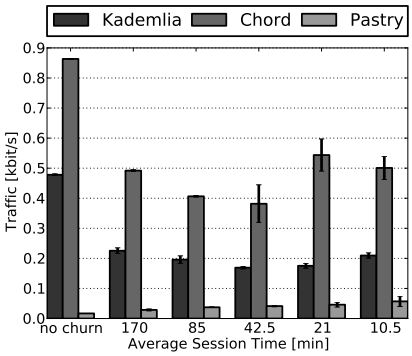


(a) Success ratio for different levels of churn.

(b) Recall for different levels of churn.



(c) Query response time for different levels of churn.

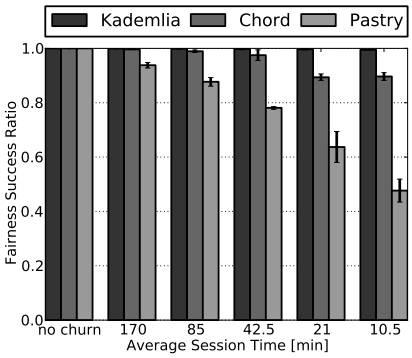


(d) Traffic per peer for different levels of churn.

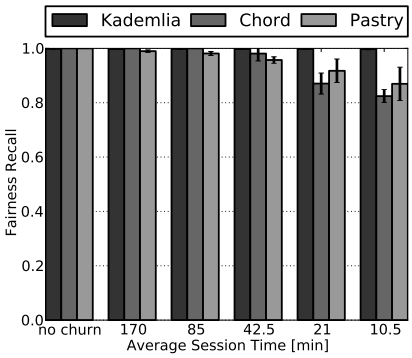
Fig. 4.2: Results for the churn benchmark.

all queries of Chord and Pastry that are resolved successfully, the recursive routing scheme provides a much better performance with low response times. From the cost point of view, Chord consumes the highest traffic, as shown in Figure 4.2d, because it actively maintains its entire routing table. Kademlia and Pastry use a passive maintenance scheme for the routing table, which detects stale peers in the routing table only during the lookup procedure at the occurrence of a timeout. The use of a passive maintenance scheme creates less traffic as unnecessary maintenance messages are avoided. In addition to the passive maintenance of its routing table, Pastry actively maintains a small leaf set of ten peers by periodically checking their online status.

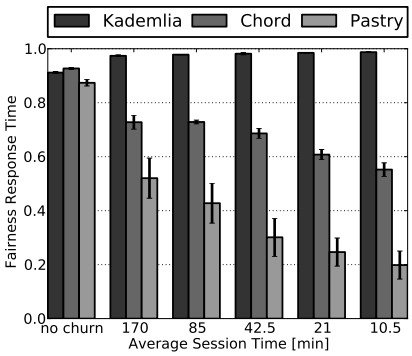
In the following, the fairness of performance and costs in the churn scenario are presented. The Jain Fairness Index is computed according to the method presented in Section 3.5.2. When examining the fairness of the three overlays, it is clearly



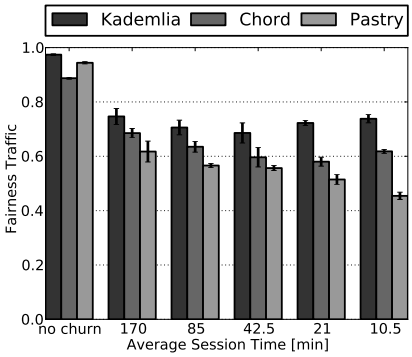
(a) Success ratio fairness for different levels of churn.



(b) Recall fairness for different levels of churn.



(c) Query response time fairness for different levels of churn.



(d) Traffic fairness per peer for different levels of churn.

Fig. 4.3: Fairness evaluation for the churn scenario.

visible that an increasing churn level leads to a more skewed distribution of performance and cost, especially for Chord and Pastry, as shown in Figure 4.3. Peers in the Pastry overlay suffer from an unfair distribution of the success ratio, as shown in Figure 4.3a. Kademlia and Chord show a fair distribution of the performance among the peers in the overlay with a fairness index close to 1. The stable behavior of Kademlia and Chord up to mean session times of 42.5 min, thus, corresponds to a fair distribution of performance among peers. Pastry shows the highest decrease in the fairness of the success ratio, which correlates to the deteriorating success ratio in terms of a decreasing session time of the peers. Considering the fairness of the recall, all peers in all three overlays receive an equal performance. This is in contrast to the response time fairness shown in Figure 4.3c. Peers in Kademlia perceive an equal response time, whereas peers in Chord and Pastry suffer from large dif-

ferences. This unequal distribution of response times can be explained by a partial failure of the two overlays, which causes peers in certain regions of the overlay to suffer from high numbers of stale contacts in their routing tables. The fairness of the operational costs in terms of traffic is shown in Figure 4.3d. Under ideal conditions all three overlays distribute the costs equally over all peers. With an increasing churn level, however, the distribution of cost becomes more skewed, resulting in a decreased fairness value. This fairness values stabilizes at a certain threshold for each overlay and does not further decrease with a higher churn level.

4.6.2 Robustness

In the following, the results for robustness are shown. The ratio of ungracefully leaving and suddenly joining peers is varied as well as the fraction of messages being dropped during transmission from a source to a target peer.

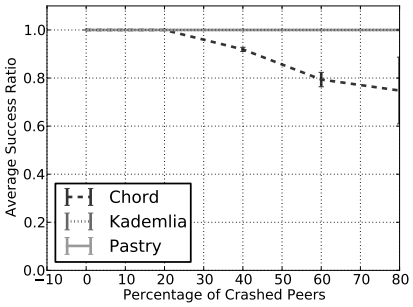
Massive Crash

The results for the massive crash scenario are shown in Figure 4.4. The plots show the results for different fractions of peers leaving the system ungracefully, ranging from 0 to 90% of the peers. For the runs with a leave ratio greater than 0%, the measured success ratio, recall, response time, and traffic right after the crash are shown. With an increasing ratio of peers suddenly leaving the overlay, the success ratio and recall of Chord is dropping rapidly, as shown in Figure 4.4a and 4.4b. In contrast, Kademlia and Pastry show a stable success ratio and recall of close to one for all ratios of peers leaving the overlay.

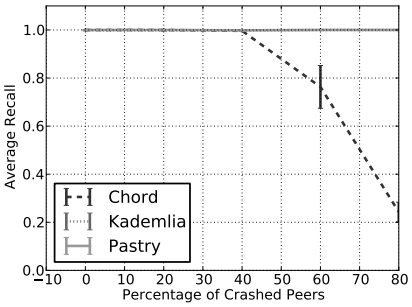
Furthermore, with a higher ratio of peers leaving the overlay, the response time of Kademlia increases (Figure 4.4c), because the routing tables of peers suddenly contain outdated peer contacts, which lead to timeouts during the execution of parallel lookups. Pastry and Chord show a stable response time, which is slightly decreasing as routing with fewer peers in the overlay performs faster. The traffic per peer in the overlay, as shown in Figure 4.4d, decreases with a higher ratio of peers leaving the system. With fewer peers in the system, there is less traffic due to routing table maintenance, and fewer hops are needed to reach a target peer, resulting in decreased traffic per peer.

Massive Join

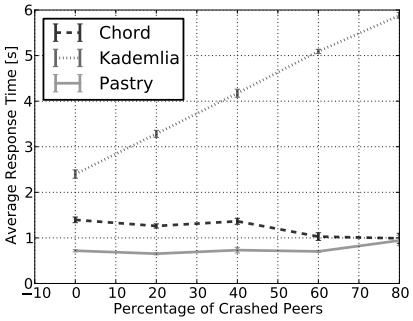
Similar to the massive leave scenario, Figure 4.5 shows the values for the success ratio, recall, response time, and traffic right after the massive join of peers took place. Figure 4.5a and 4.5b show the success ratio and recall for fractions of new



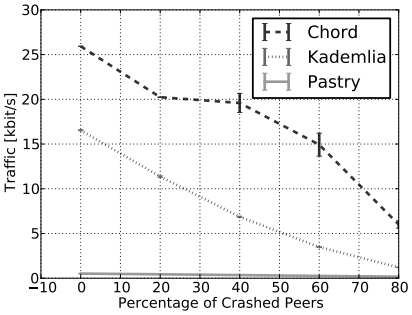
(a) Success ratio of the SUTs after the massive leave of peers.



(b) Recall of the SUTs after the massive leave of peers.



(c) Query response time of the SUTs after the massive leave of peers.



(d) Traffic of the SUTs before the massive leave of peers.

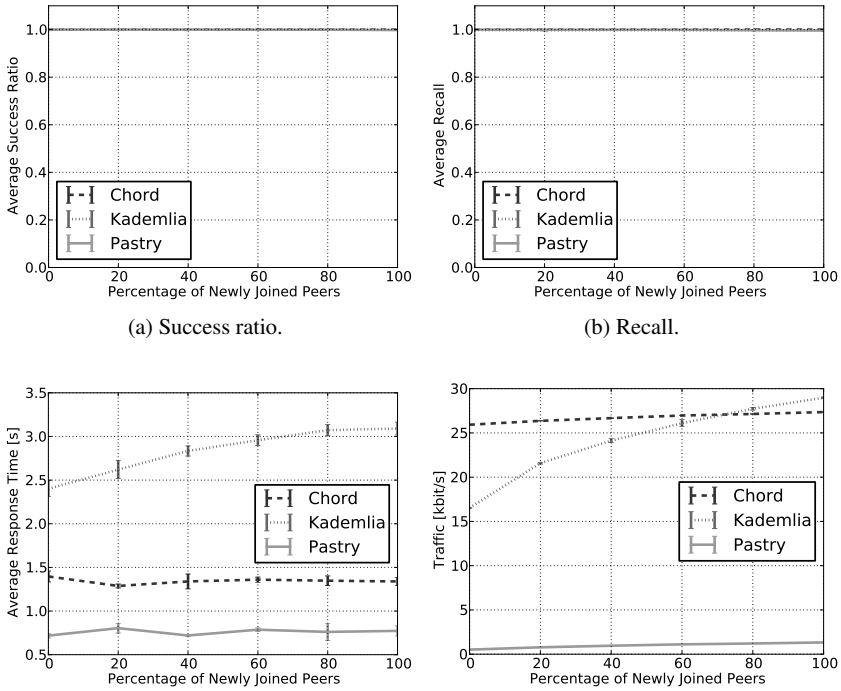
Fig. 4.4: Results for the massive crash scenario captured directly after the massive crash of the peers

peers suddenly joining the system ranging from 0 to 100%. For all fractions the three overlays deliver a high success ratio and recall of one.

From the response time point of view, all three overlays show a stable behavior. Only the response time of Kademlia is slightly increasing, as shown in Figure 4.5c. The same observation can be made for the traffic measured per peer, as shown in Figure 4.5d. Only the traffic of Kademlia is increasing because with an increasing number of peers, each peer in the system maintains more peers in his routing table, which in turn results in more peers being queried during lookups.

Message Loss

In an environment with increasing message loss all three systems are capable of handling a message loss of up to five percent. With a message loss above five per-



(a) Success ratio.

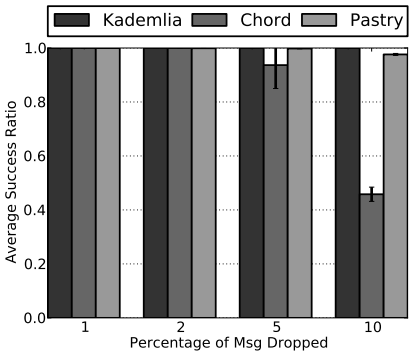
(b) Recall.

(c) Query response time.

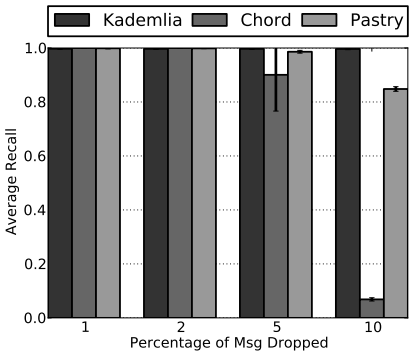
(d) Traffic per peer.

Fig. 4.5: Results for the massive join scenario

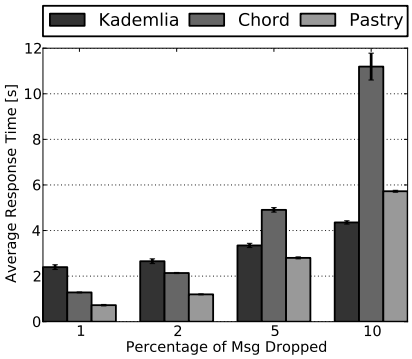
cent, the performance of Chord with respect to the success ratio and recall starts to deteriorate, as shown in Figure 4.6a and 4.6b. The overlay is not capable any longer to compensate the messages loss by simply resending messages. The loss of messages can also be recognized in the increase in the response time of all three overlays due to message timeouts taking place, as shown in Figure 4.6c. Of all three overlays, Chord shows the highest increase in the response time, which indicates that the ring-based routing concept is more susceptible to message loss. Cost-wise, the increase in the message loss results in an increase in the traffic per peer for the Chord overlay, as shown in Figure 4.6d, as Chord actively retransmits messages. In contrast to that, Kademlia does not use a retransmission scheme as it already uses the concept of parallel lookups. As long as at least one of the α parallel request messages reaches the target, the request can still be fulfilled.



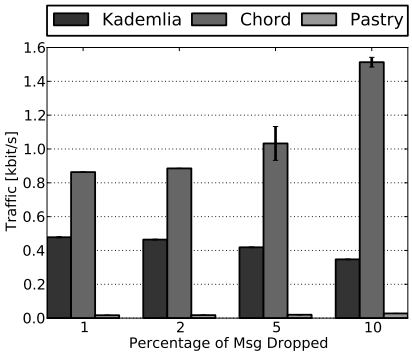
(a) Success ratio of the SUTs after the massive leave of peers.



(b) Recall of the SUTs after the massive leave of peers.



(c) Query response time of the SUTs after the massive leave of peers.

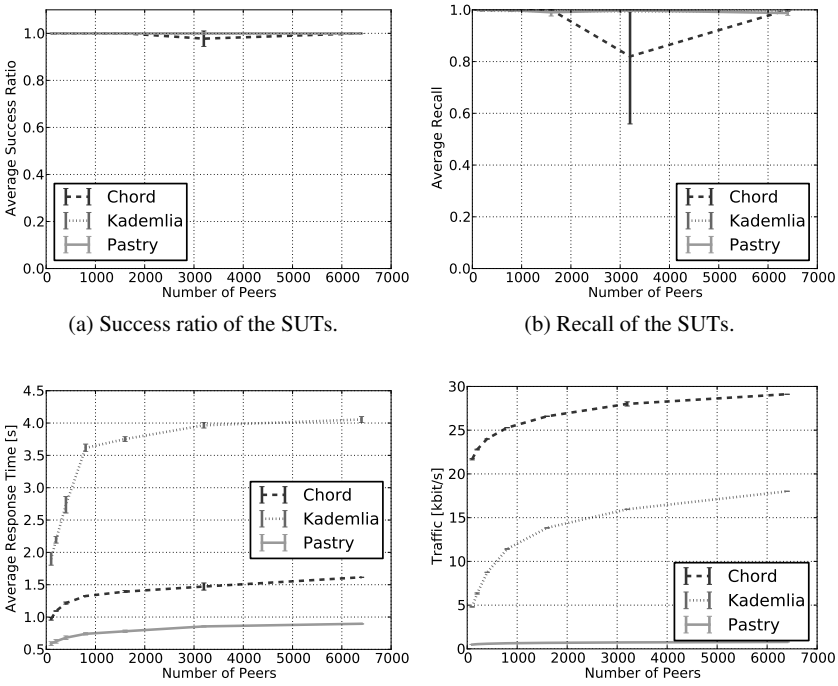


(d) Traffic of the SUTs before the massive leave of peers.

Fig. 4.6: Results for the message loss scenario

4.6.3 Scalability

Figure 4.7 shows the results of the scalability benchmark with an increasing number of peers. All systems provide an stable success ratio and recall of one, as shown in Figures 4.7a and 4.7b. Furthermore, all three systems show a logarithmic increase in the response time and traffic as shown in Figures 4.7c and Figure 4.7d.



(a) Success ratio of the SUTs.

(b) Recall of the SUTs.

(c) Query response time of the SUTs.

(d) Traffic of the SUTs.

Fig. 4.7: Results for the increasing number of peers scenario

4.7 Summary and Conclusion

In applying the benchmarking methodology presented in Chapter 3, we were able to derive a performance and cost profile of the three structured search overlays Chord, Kademlia, and Pastry. The benchmark results reveal that in terms of churn, the Kademlia overlay is much more stable due to the use of an iterative routing scheme. The recursive routing scheme used by Pastry and Chord, on the other hand, suffers from severe performance problems. Looking at the robustness, the sudden join and leave of peers can be handled well by the three overlays. Only Chord shows a significant decrease in the response time. Success ratio, response time, and traffic of all overlays remain within reasonable boundaries. With respect to the robustness against message loss, Kademlia and Pastry show a stable behavior for up to ten percent of messages being dropped. The scalability benchmark confirmed that all three overlays scale logarithmically with the number of peers.

References

[1] K. Aberer et al. “The Essence of P2P: A Reference Architecture for Overlay Networks”. In: *International Conference on Peer-to-Peer Computing*. IEEE, 2005.

[2] A. Binzenhofer. “On the Stability of Chord-based P2P Systems”. In: *Global Telecommunications Conference*. 2005.

[3] Z. Cheng et al. “Exploring Millions of Footprints in Location Sharing Services”. In: *International AAAI Conference on Weblogs and Social Media*. 2011.

[4] S. Kaune. “Performance and Availability in Peer-to-Peer Content Distribution Systems: A Case for a Multilateral Incentive Approach.” PhD thesis. Technische Universtät Darmstadt, 2011.

[5] S. Kaune, M. Wählisch, and K. Pussep. “Modeling and Tools for Network Simulation: Modeling the Internet Delay Space and its Application in Large Scale P2P Simulations”. In: ed. by James Groß Klaus Wehrle Mesut Günes. Springer, 2010.

[6] A. Kovacevic. “Peer-to-Peer Location-based Search: Engineering a novel Peer-to-Peer Overlay Network”. PhD thesis. Technische Universität Darmstadt, 2009.

[7] J. Li et al. “A Performance vs. Cost Framework for Evaluating DHT Design Tradeoffs Under Churn”. In: *Annual Joint Conf. of the IEEE Computer and Communications Societies*. 2005.

[8] P. Maymounkov and D. Mazières. “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric”. In: *Peer-to-Peer Systems*. Lecture Notes in Computer Science. Springer, 2002.

[9] K. Pussep, C. Leng, and S. Kaune. “Modeling User Behavior in P2P Systems”. In: *Modeling and Tools for Network Simulation*. Springer, 2010.

[10] A. Rowstron and P. Druschel. “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems”. In: *Middleware 2001*. Lecture Notes in Computer Science. Springer, 2001.

[11] X. Shen et al. *Handbook of Peer-to-Peer Networking*. Springer, 2009.

[12] M. Steiner, T. Najjary, and E. Biersack. “Analyzing Peer Behavior in KAD”. In: *Institut Eurecom, France, Tech.* (2007).

[13] D. Stingl et al. “PeerfactSim.KOM: A Simulation Framework for Peer-to-Peer Systems”. In: *International Conference on High Performance Computing & Simulation*. 2011.

[14] I. Stoica et al. “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications”. In: *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2001.