# Performance Evaluation of Structured Peer-to-Peer Overlays for Use on Mobile Networks

Farida Chowdhury, Jamie Furness, Mario Kolberg

Computing Science and Mathematics
Universty of Stirling, Stirling, Scotland
{fch,jrf,mko}@cs.stir.ac.uk

**Abstract:** Distributed Hash Table (DHT) based Peer-to-Peer (P2P) overlays have been widely researched and deployed in many applications such as file sharing, IP telephony, content distribution and media streaming applications. However, their deployment has largely been restricted to fixed, wired networks. This is due to the fact that supporting P2P overlays on wireless networks such as the public mobile data network is more challenging due to constraints in terms of data transmissions on cellular networks, limited battery power of the handsets and increased levels of node churn. However, the proliferation of smartphones makes the use of P2P applications on mobile handsets very desirable.

In this paper, we have analysed and evaluated the performance and efficiency of five popular DHT based structured P2P overlays (Chord, Pastry, Kademlia, Broose and EpiChord) under conditions as commonly experienced in public mobile data networks. Our results show that the conditions in mobile networks, including a high churn rate and the relatively low bandwidth availability is best matched by Kademlia and EpiChord. These overlays exhibit a high lookup success ratio and low hop count while consuming a moderate amount of bandwidth. These characteristics make these two overlays suitable candidates for use in mobile networks.

## 1. INTRODUCTION

Peer-to-Peer (P2P) communication networks have been very popular over the last decade due to their capability to facilitate information storage and retrieval among a potentially very large number of nodes while avoiding the use of central servers. Especially Distributed Hash Table (DHT) based structured P2P overlays offer an efficient routing architecture that is adaptive, self-organizing, fault tolerant, scalable and massively distributed. P2P has been used primarily for content distribution but more recently it has also been used for multicasting, distributed collaboration systems and grid computing.

P2P applications have been very successful on fixed, wired networks. With the explosive increase in popularity of smartphones the consumption of data services on the move has increased substantially, a move which makes the availability of P2P overlay networks very desirable.

On the other hand, mobile networked devices often have strict bandwidth limits and limited processing power. These limitations are typically caused by expensive bandwidth costs and restrictive battery performance. Furthermore, in a mobile network setting, the performance can be affected significantly by frequent join and leave events of mobile nodes – commonly known as *node churn*. As the proportion of mobile nodes increases in the network, many P2P overlays cannot cope and simply collapse under high churn rate [1],[2]. Furthermore, the communication link may be impacted by high packet loss rate and bandwidth fluctuations which make the mobile environment much more challenging. The mobile handset devices also have limited battery power. With P2P applications becoming increasingly popular for networked devices, P2P systems are required to minimise their network overhead to prolong device battery life while coping with high levels of node churn.

Structured P2P overlays make use of a Distributed Hash Table (DHT) to distribute index information about the shared data items across the participating peers. Each node is assigned a unique ID (e.g. a hashed IP address) and responsible for storing a range of keys which is closest to its node ID. Each data item is also assigned a file ID (e.g. the hash of a file name) and each node is responsible for a particular range of the file ID space. If a node queries a key, the network returns the node ID where the associated file with that key is stored. Therefore the nodes exchange data items based on their keys. Well-known examples of such algorithms are Chord [3], Pastry [4], Kademlia [5], Broose [6] and EpiChord [7].

Chord [3] uses consistent hash function to generate an $m$-bit identifier for each node and resource, where m is a pre-defined system parameter. Nodes are ordered to form a ring circle of modulo $2m$. Each node has a successor and a predecessor on the ring. Each node maintains a routing table which consists of two parts: the first part is a finger

table that contains the *m* entries and the predecessor of this node. The second part is a neighbour table that contains the successor list of size *r*. When a lookup is initiated, it is forwarded clockwise to its closer nodes.

Pastry [4] also consists of a circular identifier space where each node and data item are assigned a 128-bit identifier using a consistent hash function. In addition to the routing table, each node in Pastry maintains a Leaf set and a Neighbourhood set. Pastry uses prefix routing to route messages where each entry in the routing table contains the IP address of peers whose identifiers have the appropriate prefix and it is chosen according to a close proximity metric.

Kademlia [5] is another DHT based P2P overlay that assigns a NodeID in each node in the 160-bit key space and the [key,value] pairs are stored on nodes with IDs close to the key. It uses XOR-based metric topology for routing. Kademlia nodes contain lists of entries, known as buckets, which are used to send parallel lookups.

Broose [6] is based on the De-Bruijn topology that allows a distributed hash table to be maintained in a loose manner. Similar to Kademlia, it stores an association of *k* nodes instead of one to get high reliability without any node failures and uses a constant size routing table of *O(k)* contacts for allowing lookups in *O(log N)* messages exchange.

EpiChord [7] is a DHT algorithm where peers maintain a full routing table and ideally approach *O(1)* hop lookup performance compared to the *O(logN)* hop performance offered in many multi-hop networks. EpiChord is based on the Chord DHT and organized as a one-dimensional circular space where each node is assigned a unique node identifier. The node responsible for a key is the node whose identifier most closely follows the key. In addition to maintaining a list of the *k* succeeding nodes, EpiChord also maintains a list of the *k* preceding nodes and a cache of nodes. Nodes update their cache by observing lookup traffic. Therefore nodes add an entry anytime they learn of a node not already in the cache and remove entries which are considered dead. In general terms EpiChord can be thought of as Chord with a cache of extra node addresses. With a well-populated cache this results in lookup performance approaching one hop. Under high churn the performance may drop to that of Chord, *O(log N)* hops as a lower bound.

In this paper, we present a comparative analysis of these five structured P2P overlays using the OverSim [8] simulation framework under the more extreme requirements that the public mobile data networks pose.

The contributions of our paper are:

- Presentation of an EpiChord overlay model for the OverSim simulation environment and validation of the performance of the model.
- Comparative performance evaluation of Chord, Pastry, Kademlia, Broose and EpiChord, considering high levels of churn and bandwidth consumption as required in mobile networks.
- Identifying the most suitable configuration of these five DHTs to achieve the optimal performance in mobile environments.

The rest of the paper is structured as follows: Section 2 gives the related work. Section 3 compares the available P2P simulators and justifies why we have used OverSim. Section 4 presents an overview of the OverSim simulator. In section 5, EpiChord has been validated and the results have been presented after making changes to the original EpiChord model. Section 6 describes the experiment design and simulation setup. How the best parameters for each overlay have been selected is discussed in section 7. Performance evaluations are discussed in Section 8 and we summarise in Section 9.

## 2. RELATED WORK

A large number of multi-hop structured Peer-to-Peer (P2P) algorithms have been proposed [9]. These algorithms are characterized by *O(log N)* hop count. Because each overlay hop translates to potentially many hops in the underlying network, multi-hop overlays have a relatively poor latency characteristic for connecting large numbers of peers. Consequently, systems have been developed to trade-off latency for larger routing tables. However these designs lead to increased network traffic for managing larger routing tables. Thus the efficient overlay maintenance in *O(1)*-hop (one-hop) overlays is an important research question. Two techniques have emerged [9] for maintaining routing tables in overlays: active stabilization where peers have fixed communication to maintain a target routing table accuracy, and opportunistic updating where routing table maintenance depends on lookup load and available bandwidth. Active stabilization provides for higher routing table accuracy and hence lookup performance at the cost of increased maintenance traffic.

DHT based P2P overlays from the viewpoint of wired and wireless networks have received significant attention and therefore the performance have been evaluated in a number of studies [14], [15], [15],[17]. A Performance Versus Cost (PVC) framework has been presented to evaluate the cost and performance of various DHTs under churn in [14]. The authors simulated Chord, Kademlia, Kelips, OneHop and Tapestry; and showed that all of these measured DHTs can achieve reasonable performance if parameters are optimised.

In [15], Chord, Koorde, Pastry, Bamboo, Kademlia and Broose are compared based on the PVC evaluation framework [14]. While churn settings were simulated, the simulation parameters were *not* chosen to reflect the conditions of wireless networks, but are based on the conditions found in fixed wired networks. Consequently, the overlays are not configured for the conditions they are meant to operate in and thus do not yield best performance.

In [17] Chord, Kelips and Tapestry are analysed. However, the experiments were based on rather small networks of 1000 nodes. In [18], several multi-hop DHT algorithms' suitability for interpersonal communication was investigated. Desired features of DHTs were presented as well as the suitability of the algorithms for mobile networks was assessed. However, the analysis was carried out in a theoretical way and no simulation based evaluation is presented.

In [19], the performance of the Kademlia P2P system in mobile networks in the presence of different levels of churn was evaluated. They conclude that a 3-way parallel lookup together with a resource replication factor of 3 are sufficient for a robust system under churn. Besides focussing on a single overlay, the experimentation carried out was based on very small networks with about 400 nodes only.

EpiChord was originally proposed in [7]. The authors validated their model using SSFNet [20] and compared EpiChord with Chord. The performance of EpiChord has been evaluated in two different workloads: *Lookup-intensive* and *Churn-intensive*. The analysis and simulations presented in the paper show that EpiChord offers significantly better lookup hop count and success ratio performance with only slightly higher bandwidth costs than Chord by using parallel lookups and by amortizing the network maintenance costs into the lookup costs. To date, there is no further investigation of EpiChord's performance under the constraints of mobile networks. In this paper, we address its applicability for mobile networks.

## 3. REVIEW OF SIMULATORS

Before deciding on OverSim, a detailed review of other available and active P2P network simulators was carried out. A summary of these tools is provided in Table I.

**PeerSim** [21] is a Java based simulator whose main focus is to provide high scalability, with network sizes of up to $10^6$ nodes. However, this scalability comes at the cost of omitting the behavior of the underlying communication network.

**P2PSim** [22] is a discrete event simulator for P2P overlays written in C++. Models for Chord, Accordion, Koorde, Kelips, Tapestry, and Kademlia are available. However, these implementations are specific to P2PSim and do not model all features of the protocols. P2PSim has been simulated with up to 3,000 nodes using the Chord implementation. This simulator is largely undocumented and therefore hard to extend.

**Overlay Weaver** [23] is a toolkit for P2P Overlays written in Java. It has been tested with tens of thousands of nodes (their website quotes 300,000). Chord, Kademlia, Pastry, Tapestry and Koorde models are available. The simulations have to be run in real-time environments and there is no statistical output which limits its use.

**PlanetSim** [24] is a discrete event simulation framework for both structured and unstructured overlays, written in Java. It has a modular, well-structured architecture and services can be re-used for other overlays. Chord and Symphony models exist and can consist of up to 100,000 nodes. However, it provides rather limited support to collect statistics. It has a much simplified underlying network layer without consideration of bandwidth and latency costs.

**NS2** [25] is a discrete-event simulator that provides substantial support for simulation of lower layer protocols. Only one P2P protocol, Gnutella, is available in NS2. Simulations in NS2 are constructed using C++ and OTcl. It is mostly used with small networks and due to the models' complexity is generally unsuitable for large scale P2P networks.

**SSFNet** [20] is a discrete-event simulation framework written in Java and C++. This framework is built on the Scalable Simulation Framework (SSF) and uses the Domain Modeling Language (DML) to configure networks.

Chord and EpiChord have been implemented in SSFNet. There is a claim that SSFNet manages to run models with 33,000 nodes, however, the authors of the original EpiChord paper [2] and ourselves could not simulate networks with more than 10k nodes.

Table I. A comparison of available active P2P simulators

| Simulator | P2P Protocols | Network size | Language |
|---|---|---|---|
| PeerSim | Collection of internally developed P2P models | $>10^6$ | Java |
| P2PSim | Chord, Accordion, Koorde, Kelips, Tapestry, Kademlia | 3000 | C++ |
| Overlay Weaver | Chord, Kademlia, Koorde, Pastry, Tapestry and FRT-Chord | Tens of thousand | Java |
| PlanetSim | Chord, Symphony | 100,000 | Java |
| NS2 | Gnutella | N/A | C++/OTcl |
| SSFNet | Chord, EpiChord | 33,000 | Java/C++/DML |
| OverSim | Chord, Kademlia, Pastry, Bamboo, Broose, Gia | 100,000 | C++ |
| PeerfactSim.Kom | CAN, Chord, Kademlia, Gia, C-DHT, Pastry, Gnutella 0.4/0.6 | 50,000 | Java |
| D-P2P-Sim+ | Chord | 400,000 | Java |

**OverSim** [8] is an open-source P2P simulation framework for the OMNeT++ simulation environment. It provides a generic lookup mechanism and an RPC interface to facilitate additional protocol implementations. It allows large-scale simulations of simplified networks as well as complex heterogeneous underlay networks. Several P2P algorithms such as Chord, Kademlia, Bamboo, Broose, Koorde, NICE, NTree, Pastry, and GIA have been implemented in OverSim. Models can scale to over 100,000 nodes.

**PeerfactSim.Kom** [28] is a discrete event based P2P simulator environment. Its focus is on being extendable and on large scale network models. This simulator offers the potential to model different types of peer-to-peer systems including distributed CDNs, streaming applications and overlay systems. It comes with a built-in churn generator. The simulator includes models of lower layers but does not yet include TCP.

**D-P2P-Sim+** [29] is a distributed simulation environment which employs multi-threading, asynchronous message passing and distributed environment with graphical user interface. There is little information on this simulator besides a short paper and a poster. These report simulated network sizes of up to 400,000 nodes. It seems the only implemented overlay algorithm is Chord. The system is extensible and other algorithms could be implemented. Multiple computers running the simulator may be interconnected to achieve larger simulated network sizes.

Based on this study OverSim was selected for our experimentation due to its flexibility with respect to underlay characteristics, possible high scalability and a strong support of DHT based P2P algorithms. OverSim also provides an interactive GUI and real-time messages which are extremely useful for debugging models. Other surveys of P2P network simulators can be found in [26], [27].

## 4. OVERSIM

OverSim [8] is designed as a modular simulation framework, with many common overlay features implemented as part of a generic base overlay class. OverSim provides message passing using Remote Procedure Calls (RPC), and supports both iterative and recursive routing. Applications within OverSim are split into multiple tiers, allowing an application to sit on-top of another application. These applications are implemented as modules and interface with overlays through the Key-Based Routing (KBR) API [30], which represents basic capabilities common to all structured overlays. As mentioned above, OverSim provides a number of different network models, for both structured and unstructured overlays. The OverSim architecture is illustrated in Fig. 1.

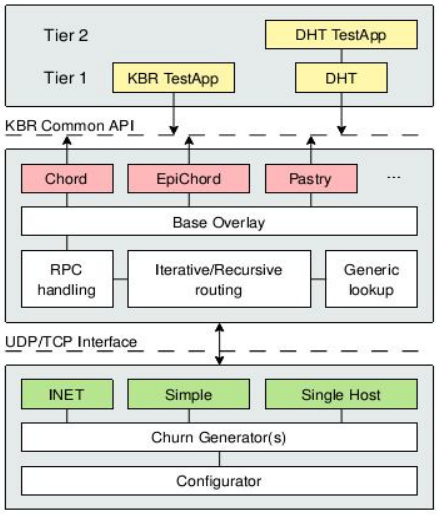


Fig. 1. OverSim architecture

At the lower layer OverSim provides multiple underlay models to allow for inclusion of specific underlay characteristics in the simulation (at a cost of scalability), or underlay abstraction for increased scalability. Using the simple model, data packets are sent directly from one node to another by using a global routing table. The INET underlay model includes simulation models for all network layers. The single host underlay allows for simulation of a single node, connected to other OverSim instances over a real network.

## 5. VALIDATION OF EPICHORD

A number of multi-hop overlay models are already available with OverSim. However EpiChord, which is a one-hop overlay, was not available in OverSim. Therefore we implemented and validated EpiChord in OverSim [31]. We made some alteration to the original EpiChord protocol while implementing it as an OverSim module. These are detailed below.

### 5.1 Node Join Protocol

In the original EpiChord algorithm, upon receipt of a join request a node will instantly update their predecessor list and finger cache to include the joining node. In our implementation we found this was occasionally causing messages to be routed to nodes who are still in the process of joining, and hence are not yet ready to correctly handle requests. To solve this issue a three-way handshake was implemented. In our implementation the joining node will send a final acknowledgment when they are ready to handle requests, indicating they can now be safely added as a predecessor.

### 5.2 Lookup Algorithm

The OverSim framework provides modules for iterative and recursive routing offering support for parallelism. While this makes implementing overlays easier and reduces duplicated code, only certain parts of the module can be easily overwritten. This was a problem for EpiChord, primarily due to the non-linear order in which nodes are to be queried, and EpiChord's ability to check for false negative responses. To implement these features changes to the iterative routing module were made, allowing overwriting additional parts of the module with EpiChord specific code.

### 5.3 Application layer Lookups

In the original EpiChord model all lookup types (JOIN, MAINTENANCE, and APPLICATION) are included when calculating results. The KBRTestApp in OverSim only includes lookups in the results, which it has initiated (APPLICATION). We feel this is actually a more useful metric for anyone wishing to build on-top of EpiChord, so we instead recalculated the results from the original model using only APPLICATION lookups. A comparison of the average path lengths can be seen in Fig. 2; the other metrics remained unchanged.

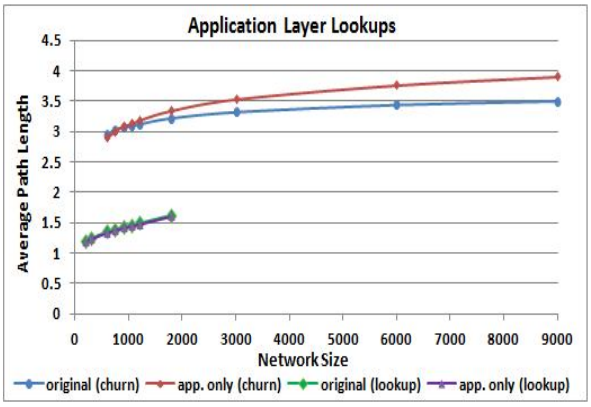


Fig. 2. Comparison of average path length with APPLICATION lookups only vs. all lookup.

In [14], authors proposed two generic classes of workloads: *lookup intensive* and *churn intensive*. These metrics were adopted by the EpiChord authors for experimentation. For the purposes of validating our model, we also adopt these two metrics. In the *lookup intensive* workload, node lifetimes are exponentially distributed with a mean of 10 minutes, with each node performing a lookup on average every 0.5 seconds. In this scenario the background maintenance traffic is negligible compared to the active lookup rate. In the *churn intensive* workload, node lifetimes are again exponentially distributed with a mean of 10 minutes, however this time each node only performs lookups on average every 100 seconds. In this scenario, lookups arising from node joins and cache maintenance substantially outnumber the active lookups.

Fig 2 shows that the average path length remains unchanged for the *lookup intensive* workload. This is to be expected, as the lookup intensive workload is dominated by APPLICATION lookups. In the churn intensive workload we see a rise in average hop count as the network size increases; this is because the results in the original EpiChord paper were dominated by JOIN and MAINTENANCE lookups, which tend to be for closer keys.

### 5.4 Fixing the lookup parameter, *p*

The original EpiChord paper [7] stipulates that a maximum of *p* lookups should be initiated. However in the source code of the original model we encountered that in many cases, *p+1* parallel requests were generated. Results comparing the average path lengths and success rates when *p=1* can be seen in Fig.3.

From these results we observe a rise in average path length, and a small decline in lookup success rate, for both workloads. We also observe a drop in the size of nodes cache tables, which increases with the network size. This is to be expected, as fewer queries are dispatched and hence fewer new nodes are discovered.

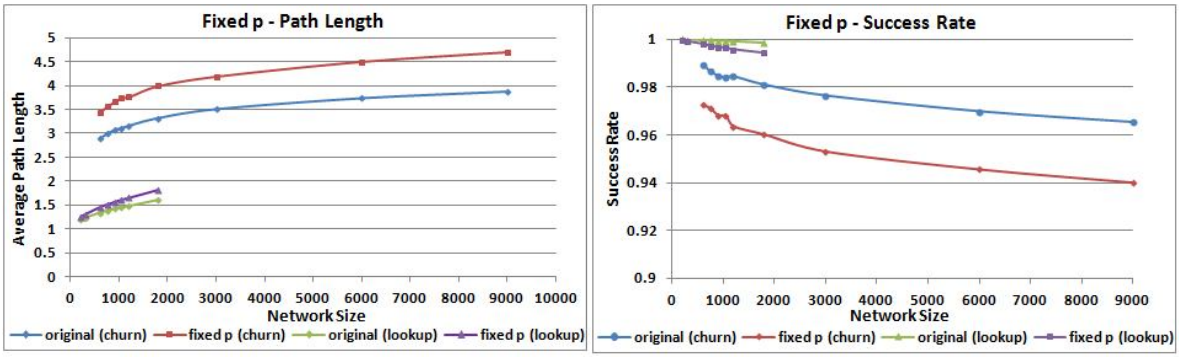


Fig. 3. Average path length and success rate with fixed *p*.

## 6. EXPERIMENTAL SETUP

In order to evaluate the suitability of overlay algorithms for use in mobile networks, a set of key requirements and performance metrics need to be identified. A set of desired features of DHTs from the viewpoint of mobile networks was proposed in [18]. Below, we briefly summarise the key requirements and then the performance metrics used in our study. The study focusses on the following overlays: Chord, Pastry, Kademlia, Broose and EpiChord.

Key requirements:

- **Suitability for Mobile Devices:** Mobile devices connected to mobile networks will be the primary devices connecting to the P2P network. The network connections used by these devices offer lower bandwidth than for wired nodes. Thus a suitable DHT should require as little bandwidth as possible.
- **Delay in DHTs:** Generally, mobile connections offer slower transmission speeds than wired connections. Thus minimizing the number of hops for each lookup will minimize the delay for lookup operations in DHTs.
- **Robustness:** A DHT suitable for use in mobile networks should be able to handle the higher degree of node churn experienced in such environments. Node churn may be introduced due to variation in signal strength, or using the network connection for voice services. A key metric to indicate the robustness of a DHT is its lookup success ratio. The DHT should achieve an acceptable lookup success ratio during periods of high churn.

Performance Metrics:

- **Mean Maintenance Traffic Load:** The mean number of maintenance bytes sent per second by each node. This metric will be considered as a parameter for bandwidth consumption in this paper.
- **Hop Count:** The mean number of overlay hops to send a message from a source to a destination node.
- **Lookup Success Ratio:** The percentage of successful lookups in the overlay.

### 6.1 Simulation Setup

Experiments have been carried out using a 10,000 nodes network. The level of churn is simulated by various mean lifetime values ranging from 100 seconds to 1000 seconds. The simulations employ OverSim's Lifetime Churn model with Weibull distribution. A shorter lifetime means a higher level of churn. The Weibull distribution has been shown to model churn in P2P overlays much more accurately [32]. Each configuration was repeated 5 times, and results were averaged. The overlays have been configured to use iterative routing.

## 7. MOST SUITED PARAMETER SELECTION

The overlay evaluation process includes two simulation steps. During the first step, the values of key parameters for each overlay are determined in order to yield the best possible performance under churn. Thus this step tunes the parameters to yield an optimal configuration for each overlay. For this step we have investigated the performance at 100s and 1000s average node lifetime. In the second step, the best performing configurations of all the overlays are compared with each other to identify the best performing overlay under churn. Table II summarizes the parameters and their values for Chord, Kademlia, Pastry, Broose and EpiChord. More details on the performance of the overlays using these parameters are provided in the following subsections.

Table II. Simulation Parameters

| P2P Protocols | Parameters | 1st step | 2nd step |
|---|---|---|---|
| Chord | Stabilize Delay (sec) | 5, 10, 20, 30, 40, 50, 60, 120 | 20 |
| | Fix-finger Delay (sec) | 30, 60, 90, 120,180, 240, 300 | 30 |
| | Successor List Size | 4, 8, 16, 32 | 8 |
| | CheckPredecessor delay(s) | 5, 10, 30, 60 | 5 |
| | Size of Extended Finger Table | 0, 1, 4, 8, 16 | 0 |
| Kademlia | Bucket Size, k | 4, 8, 16, 32, 40, 64, 72, 80 | 8 |
| | Number of Siblings, s | 2, 4, 8, 16, 32 | 2 |
| | Number of Bits, b | 1, 2, 4, 6, 8 | 1 |

| P2P Protocols | Parameters | 1st step | 2nd step |
|---|---|---|---|
| | Lookup Redundant Nodes | 1, 2, 4, 8, 16, 20 | 8 |
| | Bucket Refresh Interval (sec) | 100, 600, 1000,3000, 5000 | 1000 |
| | No. of Parallel Lookups | 1, 2, 3, 4, 5 | 3 |
| Pastry | Bits per Digit | 1, 2, 4, 6 | 4 |
| | Number of Leaves | 4, 8, 16, 32 | 8 |
| | Lookup Redundant Nodes | 2, 4, 8 | 4 |
| | Number of Neighbors | 0, 2, 4, 8, 16 | 0 |
| Broose | Bucket Size | 4, 8, 16, 32 | 8 |
| | Shifting Bits | 2, 3, 4 | 2 |
| | Bucket Refresh Interval (sec) | 30, 60, 120, 180, 300, 600 | 30 |
| | No. of Parallel Lookups | 1, 2, 3, 4, 5 | 3 |
| EpiChord | Stabilize Delay (sec) | 10, 20, 60, 100 | 60 |
| | Successor List Size | 2, 4, 8 | 4 |
| | No. of Parallel Lookups | 1, 2, 3, 4, 5 | 3 |

### 7.1 Chord

For Chord the key parameters are the stabilization delay, fixfinger delay, successor list size, check predecessor delay and extended finger table size. Fig. 4 shows the results for *success ratio* and *bandwidth consumption* with varying values for these simulation parameters.

In Chord, each node learns about newly joined nodes and updates its successors and predecessor after a stabilisation delay. For step 1 in our experimentation, we have varied the stabilisation delay between 5s and 120s. Fig. 4(a) shows that a low stabilisation delay improves the success ratio but also increases the bandwidth requirement. To compromise, a value of 20s was chosen.

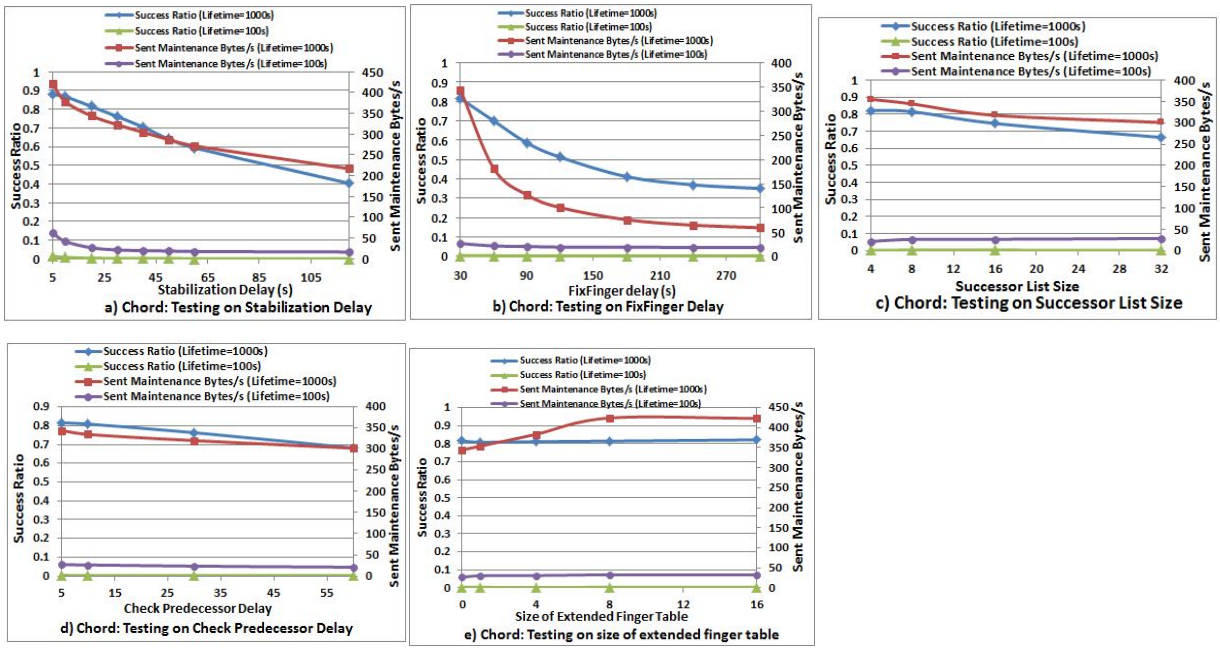


Fig. 4. Chord: Selecting best parameters according to a) stabilization delay, b) fixfinger delays, c) successor list sizes, d) check predecessor delays and e) extended finger table sizes.

In Chord, finger (routing) tables are updated in fixed intervals (fix finger delay). The fix fingers interval was varied between 30 and 300 second. Fig. 4(b) shows that a lower value improves the success ratio. We have selected a value of 30s to get the best possible lookup success ratio at a moderate level of bandwidth usage.

Results when varying the successor list size and the check predecessor delay are depicted in Fig. 4(c) and 4(d) respectively. A successor list size of 8 and the check predecessor delay of 5s were chosen to maintain a reasonable success ratio while minimizing the maintenance traffic.

Fig. 4(e) shows the effect of using extended finger tables. Based on these results, we have not increased the size of the extended finger table. We found that any increases lead to more traffic with no significant improvement in performance. Overall, we note the very poor lookup performance of Chord at high churn. We revisit this fact in the second simulation step when comparing the different overlays against each other.

### 7.2 Pastry

In Fig. 5, key parameters of Pastry are evaluated. Fig. 5(a) shows the effect of different numbers of leaf nodes. In the low churn scenario (1000s), there is no significant performance difference. In the high churn scenario (100s) we observed the highest success ratio when using 8 leaf nodes. Hence the number of leaf nodes of 8 was selected. The results of varying levels of bits per digit are shown in Fig. 5(b). Using higher number of bits can improve the success ratio as well as hop count (Fig. 5c) but it increases the maintenance cost. Considering the high churn scenario results, we have selected 4 bits per digit. In Fig. 5(d), we simulated Pastry with neighbour sets and without neighbour sets and did not notice any major improvements in success ratio (and bandwidth consumption). Therefore we decided not to use neighbour sets. In Fig. 5(e), the effects of altering the number of redundant nodes are shown. Higher values of redundant nodes can improve the success ratio but also cause an increased amount of maintenance traffic. Therefore a fixed value of 4 was chosen to balance between a good lookup success ratio and maintenance traffic load.

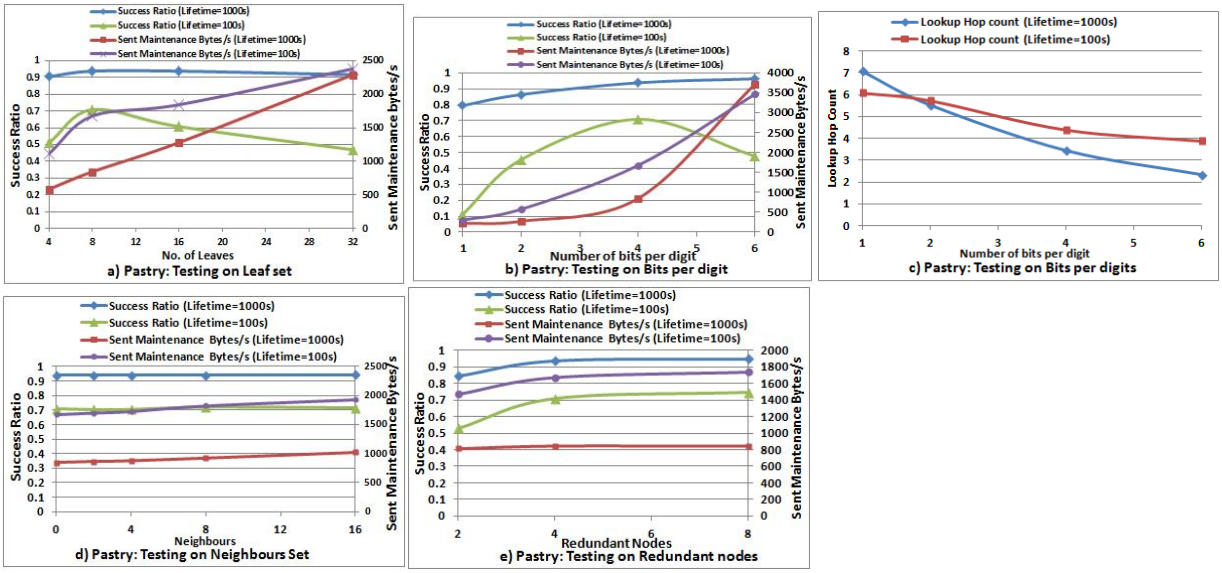


Fig. 5. Pastry: Selecting best parameters according to a) Leaf set, b) Bits per digits, c) No. of redundant nodes and d) No. of Neighbours.

### 7.3 Kademlia

Fig. 6 illustrates the *success ratio* and *bandwidth consumption* for different parameters of Kademlia. The number of buckets ($k$), number of siblings ($s$), number of redundant nodes ($r$) and number of bits per digit ($b$) were varied in Fig. 6(a), 6(b), 6(c) and 6(d) respectively. To achieve the best performance, we evaluated the traded-off between a higher success ratio and the increased maintenance traffic.

Consequently, we selected the value of 8 for $k$, 2 for $s$, 8 for $r$ and 1 for $b$. In Fig. 6(e), the bucket refresh interval was altered. As shown in the figure, an interval of 1000s is sufficient to keep the buckets consistent. As Kademlia supports parallel lookups, we investigated the effect of parallel lookups in Fig. 6(f). In high churn environments, parallelism lookups can increases the success ratio significantly. Therefore we selected a value of 3 for lookup parallelism with a moderate increase in bandwidth.

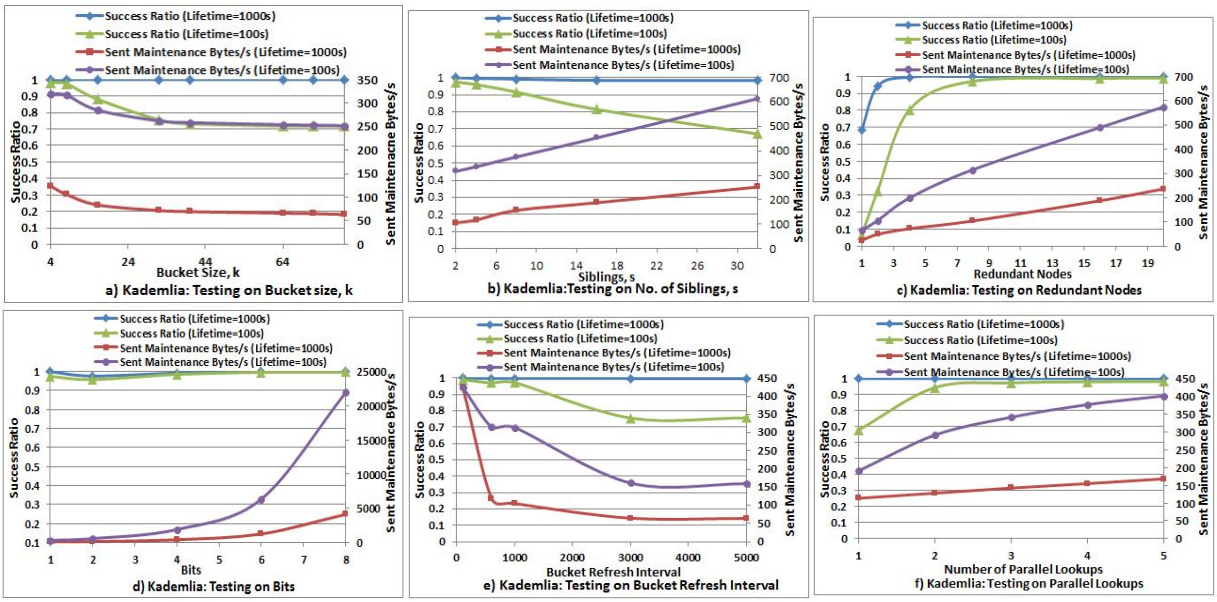


Fig. 6. Kademlia: Selecting best parameters according to a) Bucket size, b) No. of siblings, c) No. of redundant nodes, d) Bits per digits, e) Bucket refresh interval and f) No. of parallel lookups.

### 7.4 Broose

Fig. 7 shows the results for different parameters of Broose. The bucket size, *k* and bucket refresh interval were varied in Fig. 7(a) and 7(b) respectively. Fig 7(a) shows that the success ratio is improved with larger buckets; however also the maintenance traffic increases drastically. Therefore a value of 8 for *k* is appropriate. In terms of the bucket refresh interval, a value of 30s was selected in order to get the best performance as well as moderate level of maintenance traffic under high churn. Shifting more than one bit at each routing step clearly improves the success ratio, however, also increases the maintenance traffic. Thus we have selected a value of 2 bits (Fig. 7(c)). Increasing parallel lookups does not show any significant lookup performance improvement (Fig. 7(d)). Hence the value of 3 has been selected.

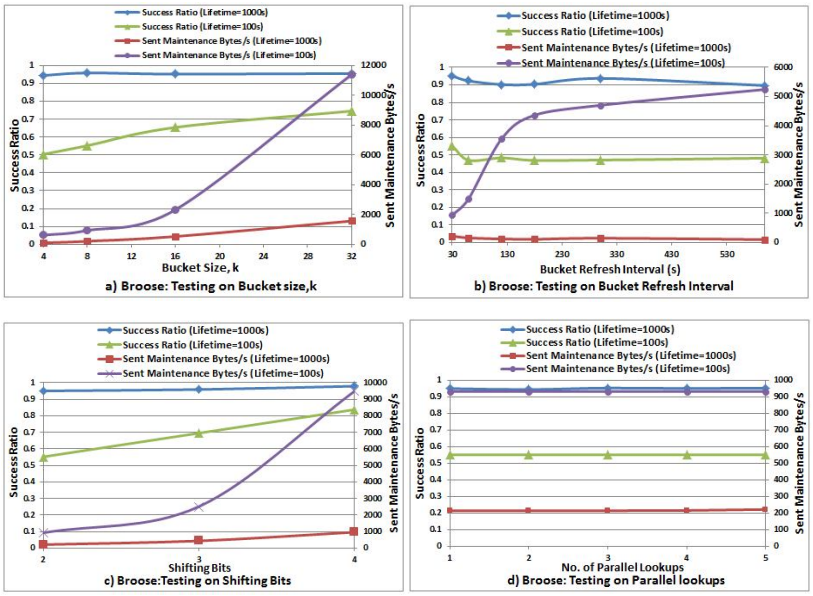


Fig. 7. Broose: Selecting best parameters according to a) Bucket size, b) Bucket refresh interval, c) Shifting bits and d) No. of Parallel lookups.

### 7.5 EpiChord

Fig. 8 reveals the results of *success ratio* and *bandwidth consumption* for EpiChord. The stabilization delay and successor list size were altered in Fig. 8(a) and 8(b) respectively. Increasing the stabilization delay does not affect the performance of EpiChord regarding the success ratio and bandwidth consumption; therefore a value of 60s was selected. The successor list size of 4 is a fair choice to compromise success ratio performance and maintenance traffic load. Fig 8(c) shows the effect of parallel lookups on success ratio and maintenance traffic. However, in EpiChord, the parallel lookup does not substantially affect the success rate and the maintenance traffic. Rather, the hop count (path length) to the destination node is improved. Also as the parallelism applies to lookup messages only, considering the lookup traffic is a better measure of cost. In Fig. 8(d), we show the lookup hop count and lookup traffic for varying levels of parallelism. Based on these results we have opted for a level of parallelism of 3.

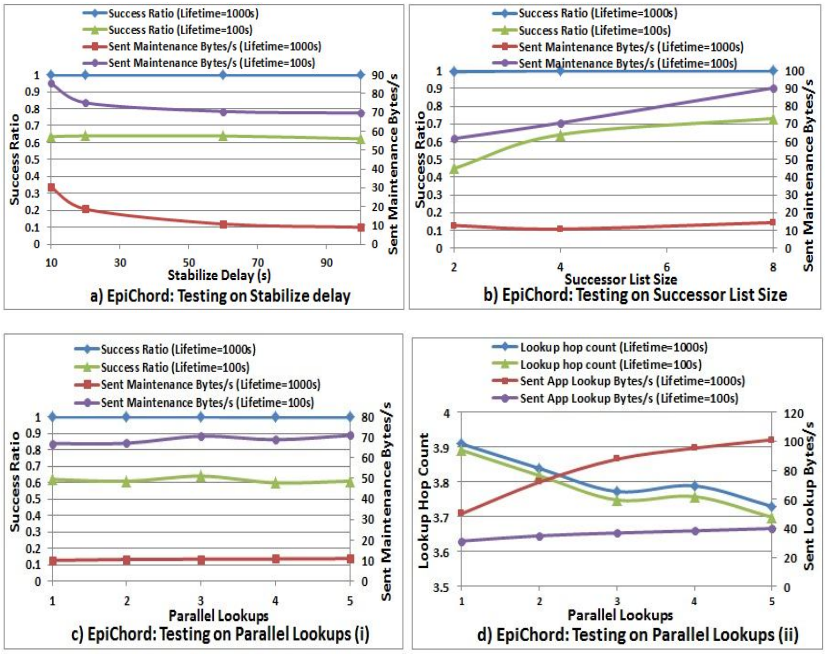


Fig. 8. EpiChord: Selecting best parameters according to Stabilize delay, b) Successor list size, c) Parallel lookups(i) and d) Parallel lookups (ii).

## 8. PERFORMANCE EVALUATION

In the following, we present the simulation results of the chosen P2P overlays with the selected parameters from the previous step, and evaluate the performance under high level of churn according to lookup success rate, hop count and bandwidth consumption.

### 8.1 Effects of Churn

In this experiment the performance of the overlays under varying levels of churn was observed. Each node performs a lookup every 60 seconds in all overlays. Results in Fig. 9a show the lookup success rate under varying levels of churn. It is evident from the figure that Chord's performance is heavily affected by high levels of churn. For a node lifetime of 100 sec the lookup success ratio collapses to a mere 2%. While this increases to about 80% as the node lifetime increases to 1000 sec, it is evident Chord is not a good candidate for a network consisting of mobile nodes. One likely reason for Chord's poor performance is that Chord does not immediately correct its successor and predecessor pointers of all relevant nodes as new nodes join, but waits for the periodic stabilization process to update the pointers. Therefore under high levels of churn when nodes join and leave the network at a very high rate, routing table entries are not updated sufficiently frequently.

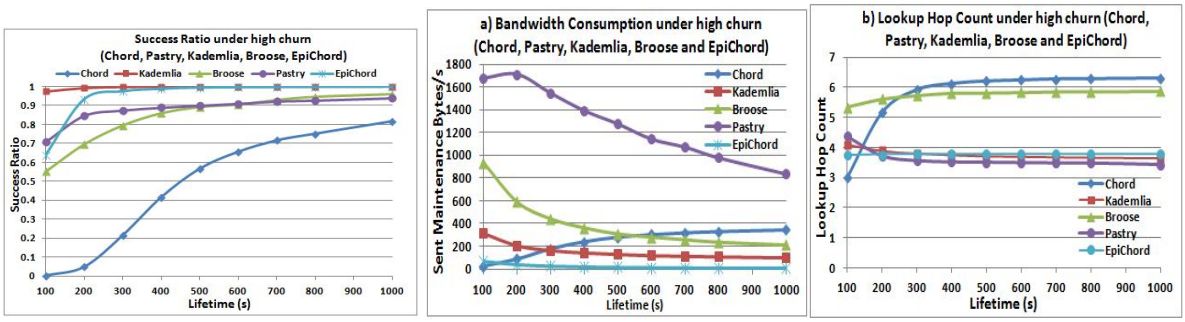


Fig. 9. a) Lookup Success Ratio b) Bandwidth consumption and c) Lookup hop count of Chord, Kademlia, Pastry, Broose and EpiChord operated on 10,000 networks under various level of churn.

In our experiment, Broose exhibits around 55% of lookup success ratio at 100 sec of node lifetime (this increases to about 95% at 1000 sec of node lifetime). However, when nearly half of lookups fail in high churn networks, Broose also does not seem well suited for such environments. This is emphasized by the fact that Broose has comparatively high bandwidth costs in high churn situations (Fig 9b).

In Pastry, the lookup success ratio is 70% at 100 sec of node lifetime, increasing to around 93% at 1000 sec of node lifetime. While this performance is acceptable, this comes at the price of the highest maintenance cost (with quite a large margin, Fig 9b). Thus Pastry has to be ruled out due to this cost.

Kademlia exhibits a 97% successful lookup rate in the high churn environment (100 sec mean node lifetime). Kademlia sends parallel lookup requests to avoid timeout delays from failed nodes. Also Kademlia updates routing table entries from data attached to lookups rather than requiring separate maintenance requests. These characteristics paired with a modest bandwidth usage and modest lookup count (4 for 10,000 nodes network) as shown in Fig 9b) and c) respectively, make Kademlia well suited for mobile environments.

Finally, EpiChord, an overlay which can approach one-hop performance, approached 99% success ratio under lower levels of churn (lifetime=1000s). In high churn, the overlay performed less favourably with a lookup success rate of about 63%. This increases to a more usable 97% as the mean lifetime approaches 300 seconds. Like Kademlia, EpiChord uses reactive routing state maintenance employing lookup response messages to carry routing table update information. EpiChord also uses parallel lookup messages. As we show separately, EpiChord can achieve an even better hop count performance at the cost of an increased number of lookups [33].

### 8.2 Bandwidth Consumption

Figure 9(b) shows the results on the required bandwidth for the selected overlays under varying levels of churn. In high churn environments when the node lifetime is 100s, Chord and EpiChord have the lowest bandwidth cost using 26 bytes/s and 70 bytes/s respectively. However, as already discussed, Chord's lookup performance is very poor under these conditions. Pastry consumes the highest bandwidth which is 1674 bytes/s at this level of high churn. This is due to its expensive joining process and reactive maintenance for the routing table, leaf set and neighbour list. Broose also requires significant bandwidth at high churn levels (934 bytes/s per node at the node lifetime of 100s). Kademlia consumes higher traffic (316 bytes/s) than Chord and EpiChord however less than Pastry and Broose, even under very high churn.

Unlike all other P2P overlays compared here, Chord shows an increasing amount of maintenance traffic with an increasing node lifetime. This is due to Chord periodically calling its stabilization process and updating its finger tables (routing tables). This approach increases the bandwidth requirements as nodes stay in the network for longer. The lookup hop count for all overlays is plotted in Fig. 9(c). Kademlia, Pastry and EpiChord show similar hop counts (~3.5) under varying levels of churn whereas Chord and Broose exhibit a higher lookup hop count.

### 8.3 Discussion

Based on the simulations of five popular structured P2P overlays, the following observations can be made:

- Chord's algorithm is relatively simple. However, Chord's lookup performance collapses to a mere 2% under high node churn. This is due to the inconsistency of node pointers. When a new node joins the network, Chord does not update successor and predecessor pointers of all relevant nodes immediately. It heavily relies on its periodic

stabilization process. Consequently, under high levels of churn, Chord's routing table entries become out of date and the lookup success ratio degrades drastically.

- Pastry requires considerably higher bandwidth under high churn than the other overlays. This is because Pastry nodes use parts of other nodes' routing tables to build their own routing table. Furthermore, Pastry's complex algorithm for optimizing routing tables requires additional bandwidth. Pastry nodes detecting node failures have to repair their states by gathering information from other nodes. This affects the overall performance of success ratio and lookup hop count. Thus it is not a good candidate for mobile environments.
- Broose exhibits an average result for lookup success but consumes more bandwidth than Chord, EpiChord and Kademlia. It also requires a higher lookup hop count than the other overlays. This is because Broose has a very exhaustive and expensive node join process. When a node joins the network, the joining node builds up its routing table using queries to other nodes for their routing table entries. Broose needs a large bucket for redundancy and cannot reduce the number of routing steps.
- Kademlia shows the best result in terms of lookup success ratio at very high churn paired with a reasonable use of bandwidth. EpiChord shows moderate bandwidth consumption and a good lookup success performance even under high churn. Note that EpiChord's performance can be improved further by introducing additional lookups to the network [33]. The requirements of mobile networks where the churn rate is expected to be high and the bandwidth availability low (or expensive) matches well with Kademlia and EpiChords' high lookup success ratio, reasonable amount of bandwidth consumption and low hop count and thus make these two candidates well suited to be used in mobile networks. Both Kademlia and EpiChord use *opportunistic maintenance* mechanisms to keep their routing tables up-to-date. In opportunistic maintenance, a node attaches routing table data to a response lookup message. The receiver then updates its routing table with this information - adding new node entries and removing node entries which are considered dead [9]. This kind of maintenance saves a bandwidth as it reduces the need for dedicated messages to update routing table entries.

  With increasing node churn the demand for routing table accuracy increases. In such situations and depending on the number of lookup messages sent in the network, opportunistic maintenance alone may not be adequate and nodes may need to employ sending additional lookup messages to receive more routing table updates. Both Kademlia and EpiChord use lookup parallelism to improve their routing table accuracy and lookup efficiency.

## 9. SUMMARY

Distributed Hash Table (DHT) based structured P2P overlay networks offer an efficient routing architecture that is adaptive, self-organizing, fault tolerant, scalable and massively distributed. Such an overlay network provides a suitable substrate for developing distributed applications. It would be desirable to offer P2P applications also on devices connected to the public mobile networks. Nevertheless there are concerns regarding the performance and efficiency of these overlays in mobile environments. Therefore we have evaluated the performance of four multi-hop structured P2P overlays: Chord, Pastry, Kademlia and Broose and one one-hop structured P2P overlay EpiChord in OverSim in the presence of high churn and investigated their suitability for mobile networks.

The simulation results suggest Kademlia as the most appropriate P2P overlay to implement on the mobile network according to lookup success ratio under high levels of churn. At the same time Kademlia consumes moderate level of bandwidth. Similarly, EpiChord also achieves a high success ratio while consuming the least amount of bandwidth making it also a strong candidate algorithm for mobile environments. Using opportunistic maintenance mechanism and lookup parallelism make Kademlia and EpiChord more efficient than the other overlays.

## REFERENCES


[1] J. Li, J. Stribling, R. Morris, M. Kaashoek, and T. Gil. A performance vs. cost framework for evaluating DHT design tradeoffs under churn. 24th IEEE Infocom, vol. 1, Mar. 2005, pp. 225–236.

[2] H. Pucha, S. M. Das, and Y. C. Hu. How to implement DHTs in mobile ad hoc networks. In the 10th ACM Intl. Conf. on Mobile Computing and Network, Sept. 2004.

[3] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. ACM SIGCOMM, pages 149–160, 2001.

[4] Antony Rowstron and Peter Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pages 329–350, Heidelberg,Germany, 2001.

[5] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In the 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS 02), London, UK, 2002 pp. 53-65.

[6] T. Gai and L. Viennot. Broose: a practical distributed hashtable based on the de-bruijn topology. 4th International Conference on Peer-to-Peer Computing, 2004, Aug. 2004, pp. 167–174.
[7] B. Leong, B. Liskov, and E. D. Demaine. EpiChord: Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management. *Computer Communications,* Elsevier Science, Vol. 29, pp. 1243-1259.
[8] I. Baumgart, B. Heep, S. Krause. OverSim: A Flexible Overlay Network Simulation Framework. 10th IEEE Global Internet Symposium (GI '07), May 2007.
[9] K. Dhara, Y. Guo, M. Kolberg, X. Wu, Overview of Structured Peer-to-Peer Overlay Algorithms, Handbook of Peer-to-peer Networking, Springer, 2009.
[10] L. Monnerat, C. Amorim. D1HT: A Distributed One Hop Hash Table. 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS), April 2006.
[11] A. Gupta, B. Liskov, R. Rodrigues. Efficient routing for peer-to-peer overlays. 1st Symposium on Networked Systems Design and Implementation (NSDI), 2004.
[12] J. Buford, A. Brown, M. Kolberg. Analysis of an Active Maintenance Algorithm for an O(1)-Hop Overlay. IEEE Globecom 2007.
[13] J. Li, J. Stribling, R. Morris, M. F. Kaashoek. Bandwidth-efficient management of DHT routing tables. Symposium on Networked System Design and Implementation (NSDI) 2005.
[14] J. Li, J. Stribling, R. Morris, M. Kaashoek, and T. Gil. 2005. A performance vs. cost framework for evaluating DHT design tradeoffs under churn. 24th IEEE Infocom, vol. 1, Mar. 2005, pp. 225–236.
[15] I. Baumgart and B. Heep. Fast but economical: A simulative comparison of structured peer-to-peer systems. 8th Euro-NF Conference on Next Generation Internet. IEEE, June 2012.
[16] Harjula, E.; Koskela, T.; Ylianttila, M.. Comparing the performance and efficiency of two popular DHTs in interpersonal communication. *Wireless Communications and Networking Conference,* March 2011.
[17] Hung Nguyen Chan, Khang Nguyen Van, Giang Ngo Hoang. Characterizing Chord, Kelips and Tapestry algorithms in P2P streaming applications over wireless network. In International Conference on Communications and Electronics ICCE 2008, Hoi An, Vietnam.
[18] Jani, H. and C. Gonzalo. 2007. Evaluation of DHTs from the viewpoint of interpersonal communications. *6th Intl. Conf. on Mobile and Ubiquitous Multimedia.* ACM: Oulu, Finland.
[19] Z. Ou, E. Harjula, O. Kassinen, and M. Ylianttila. 2010. Performance Evaluation of a Kademlia-based Communication-oriented P2P System under Churn. Elsevier Journal of Computer Networks, vol 54, pp. 689-705, April 2010.
[20] The SSFNet project. Accessed 01-January-2013. Available:http://www.ssfnet.org/
[21] PeerSim P2P Simulator. Accessed 05-Jan-2013. http://peersim.sourceforge.net.
[22] P2Psim: A Simulator for Peer-to-Peer (P2P) Protocols. http://pdos.csail.mit.edu/p2psim/
[23] K. Shudo, Y. Tanaka, S. Sekiguchi. Overlay Weaver: An Overlay Construction Toolkit, Computer Communications,Vol.31, Issue2, pp. 402-412 (2007).
[24] PlanetSim: An Overlay Network Simulation Framework. http://planet.urv.es/planetsim
[25] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/
[26] A. Brown and M. Kolberg. Tools for peer-to-peer network simulation. Internet-Draft Version 00, IETF, January 2006.
[27] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai. A Survey of Peer-to-Peer Network Simulators. In the 7th Annual Postgraduate Symposium, Liverpool, UK, 2006.
[28] D. Stingl, C. Groß, J. Rückert, L. Nobach, S. Kovacevic, R. Steinmetz. PeerfactSim.KOM: A Simulation Framework for Peer-to-Peer Systems. Intl. Conf. on High Performance Computing & Simulation (HPCS), 2011.
[29] S. Sioutas, K. Tsichlas, G. Papaloukopoulos, Y. Manolopoulos, E. Sakkopoulos. A novel Distributed P2P Simulator Architecture: D-P2P-Sim. ACM Intl. Conf. on Information and Knowledge Management (CIKM), Hong Kong, 2009
[30] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz. Towards a Common API for Structured Peer-to-Peer Overlays. Peer-to-Peer Systems II, 2735:33–44, 2003.
[31] J. Furness, F. Chowdhury, M. Kolberg. An Evaluation of EpiChord in OverSim. 5th International Conference on Networks & Communications (NETCOM - 2013), Chennai, India.
[32] 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.
[33] F.Chowdhury, M.Kolberg, Performance Evaluation of EpiChord under High Churn. 8th ACM PM2HW2N Workshop – 2013, Barcelona, Spain.