Performance Evaluation of Lookup Mechanisms in Structured and Unstructured P2P Networks under the Impacts of Churn

---

A thesis submitted in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science

in the University of Canterbury

by

Adam Chang

Supervisory Committee:

Prof Krzysztof Pawlikowski	Supervisor
Prof Harsha Sirisena	Co-Supervisor
Dr Allan McInnes	Associate Supervisor

Examining Committee:

Prof. Franco Davoli, University of Genova, Italy
Prof. Krzysztof Pawlikowski, University of Canterbury, New Zealand

University of Canterbury

2010

Abstract

This Master thesis investigates the performance of the lookup mechanisms in structured and unstructured Peer-to-Peer (P2P) networks under the impacts of churn. The considered performance metrics include the distance of matched lookups, bandwidth consumption and latencies. We have selected to study Chord, Kademlia and GIA, because these selected P2P networks possess distinctive characteristics. Chord is one of the first structured P2P networks that implements Distributed Hash Tables (DHTs). Kademlia is a more recent structured P2P network, with the XOR mechanism for improving distance calculation. GIA is an unstructured P2P network which has been proposed as an alternative to structured P2P networks based on Distributed Hash Tables. It implements a dynamic topology adaptation algorithm, flow control and biased random walks. The hypothesis was that unstructured P2P network such as GIA could operate better than structured networks such as Chord and Kademlia. Our research involved a series of simulation studies using two network simulators OverSim and OMNeT++. The simulation was also used to investigate the influence of iterative and recursive routing methods, scalability, and resiliency issues, on the performance of the file lookup mechanisms in the selected P2P networks. The results show that, in considered scenarios, GIA performs better than structured P2P networks, especially in resolving file lookups with a match that is closer to the source peer, thus conserving bandwidth.

Table of Contents

Acknowledgements

I would like to thank my supervisors, Professor Krzysztof Pawlikowski, Professor Harsha Sirisena and Dr Allan McInnes for their teachings, patience and guidance throughout this research project.

The department IT support staff, Phil and Joffre have been very helpful in overcoming our technical difficulties in this research project.

I also would like to thank my colleagues for their support through the long hours in the office. Their suggestions on surviving through the workload have proven to be invaluable.

Finally, I would like to thank my family for their support and encouragement.

Chapter 1

Introduction

What is Peer to Peer (P2P) Computing? The Internet has encouraged global-scale distributed applications and services, such as eBay and Google, to grow and become integral components of everyday operations of modern society. These services are provided to us through the client-server paradigm, which involves a significant amount of computing resources. For example, Google search engine collects information by sending crawlers around the World Wide Web, indexing the collected data, and preparing a query interface for users. In 2005, Google reports stated that 15,000 servers are required to maintain their search engine services [17]. The client-server architecture involves significant amount of planning, deployment, daily administration, and maintenance cost. To avoid these complications, Peer to Peer (P2P) networks have emerged. P2P computing has been designed to provide information sharing, utilize better the current Internet infrastructure, and to push processes of resource discovery and retrieval to the end users.

Client-server networks are asymmetric and P2P networks are symmetric. In asymmetric networks, only the clients request data or service from the servers so they can be viewed as centralized systems. Centralized systems require multiple servers and load balancing algorithms to maintain service. Additionally, they also create single-point of failure and bottlenecks. P2P networks are decentralized, and each peer can be both the server and client pending on what tasks the peer tries to achieve. All the tasks such as network management, resource discovery and retrieval are carried out by individual peers. To achieve this, P2P networks need more complex algorithms to improve scalability, load-balancing, and to maintain to efficiency in an environment with dynamically changing number of peers known as churn.

P2P networks are classified into two major groups, structured and unstructured. Structured networks such as Chord [46] and Kadmelia [38] are implemented using distributed hash tables (DHT) which provide similar functionalities as searches in a hash

table with a (key, value) pair . In structured networks, peers follow a set of rules on how connections are established between each other, and information on routing and resource locations are distributed across peers. They are considered to be more robust, can retrieve rare lookup matches, and have good scalability [45].

Unstructured networks are classified into three groups: centralized, pure, and hybrid. Centralized unstructured networks deploy file indexing servers to resolve file lookups, and peers are responsible for content distribution. Napster [8] is an example of such a centralized network.

Peers in pure unstructured P2P networks connect to each other arbitrarily, and nodes help each other with file lookups and distribution. Pure P2P networks typically use flooding to route file lookups, and Gnutella 0.4 [9] is an example of pure P2P network. In hybrid unstructured networks, such as GIA [26], peers with fewer resources connect to peers with more resources, and depend on them to resolve file lookups.

1.1 Problem

There are performance tradeoffs in file-lookups between structured and unstructured networks. In structured networks, peers take more responsibility in managing the whole network and rely on each other to resolve file lookups. However, this type of architecture would result in increased bandwidth consumption and is vulnerable to changes in the network structure. In unstructured networks such as Gnutella 0.4 [9], file lookups adopt the flooding technique which consumes a lot more bandwidth and becomes impractical in larger network. However, such unstructured P2P network as GIA [26] has implemented a flow control mechanism to avoid flooding, and its file lookup performance could be more efficient than structured networks.

The phenomenon of peers joining and leaving a network at random is known as churn [31], [50]. Churn becomes a specific phenomenon in P2P networks, caused by the fact that there are no central entities that are aware of changes in the network, so content management belongs to individuals' responsibilities. To sustain the churn, the overlay

structure has to recover from disconnected peers and stabilize following the admission of newly joined peers. The availability of contents will greatly affect the performance of file lookups. It is important to identify its effects on performance of P2P networks. There have been many arguments for using different types of churn models, ranging from uniform distribution of lifetime durations of active peers to Weibull ([20], [40], [48]), Pareto ([30], [51]) or Poisson ([22], [36], [42]) probability distributions. Their full validation requires further research.

Another concern which we identified as crucial in studying P2P file lookups is the ability to analyze details of the underlay network, over which P2P networks operate on. The number of peers in existing P2P networks can exceed millions, so computing power of a typical computer installation is unable to produce sufficiently large samples of output data during simulation studies of large P2P systems in a feasible time. Thus, one needs to trade off the size of simulated networks and the accuracy of performance evaluation. Many research projects focused on increasing the size of the simulated network or reducing simulation time by abstracting the routing details, e.g. [49].

There have been arguments that structured P2P networks that implement DHT are better than unstructured P2P networks [26]. In our research project, we have chosen to study Chord and Kademlia, to represent the structured P2P networks, and GIA, to represent unstructured P2P networks. Gnutella 0.4 with flooding search method has been analyzed in [49] so it will not be included in our research project.

1.2 Methodology

To investigate the effects of churn and evaluate the performance of lookup mechanism in Chord, Kademlia and GIA, we have performed simulations using OverSim[1]. OverSim is an overlay simulator that is integrated with the OMNeT++ network simulator [2], and the INET framework [10]. Our selection of these simulators was preceded by a survey of simulators which could be used for studying P2P networks. The survey can be found in Appendix C.

1.3 Research Goals

The thesis aims to provide a comparative study of performance of file lookup mechanisms in three different P2P networks: two structured networks Chord and Kademlia versus unstructured networks such as GIA.

Results from this research will give us a better understanding of the impact of churn on the selected P2P networks, and the file lookup performance of GIA, as compared to Chord and Kademlia. Chord is a well known P2P network implemented with DHTs, has the ability to locate resources, and remains robust as networks grow. Kademlia is another P2P network implemented with DHTs, and it adopts the XOR mechanism to improve the efficiency in resource discovery. Our goal was to show in quantitative way that GIA is a more efficient P2P network than its structured counterparts, despite that would be against a popular opinion that unstructured P2P networks without central entity management cannot be efficient, especially as their size increases.

1.4 Thesis Structure

Chapter 2 of this thesis introduces the architecture of structured and unstructured P2P networks, with placing the focus on Chord, Kademlia and GIA. This chapter also includes two literature reviews on performance of file lookups in P2P networks. Chapter 3 presents the experiment design of the performance modelling of lookup mechanisms in P2P networks. Chapter 4 presents the details of performance evaluation of file lookups in Chord, Kademlia, and GIA. Results and discussion are also included in this chapter. Chapter 5 concludes our research.

Chapter 2

Survey of P2P networks

In this chapter we consider the classifications of structured and unstructured P2P networks. The former class include Chord and Kademlia, the latter is represented by GIA. The architecture and file lookup mechanism of these networks are presented in some detail. Later in the chapter, we look into recent research on performance evaluation of P2P networks.

P2P networks share distributed resources without the assistance from central servers [45]. These networks can be categorized into structured and unstructured networks. There are three types of unstructured P2P networks which are centralized, pure, and hybrid. In centralized unstructured P2P networks, all peers rely on central indexing servers to resolve file lookups. An example of centralized unstructured P2P networks is Napster [8]. Napster was created in 1999 and attracted millions of users because of its ease on distributing resources across different networks. The architecture of centralized networks creates problems such as network bottlenecks and single point of failure. To overcome these problems Nullsoft implemented a pure P2P network in 2000 [45] known as Gnutella 0.4 [9]. In pure P2P networks peers are connected to each other and rely on each other to resolve file lookups. The lookup mechanism in Gnutella is not efficient as it floods search request across a number of peers when a peer initiates a search. The idea of hybrid P2P networks such as GIA was proposed in 2003. In hybrid P2P networks peers are connected to high capacity peers and rely on them to resolve file lookups. The lookup mechanism in GIA can only initiate search requests after receiving tokens from high capacity peers. At around the same time as hybrid architectures were proposed, structured P2P networks were also released. Structured P2P networks such as Chord and Kademlia adopt Distributed Hash Tables (DHTs) to provide distributed indexing, scalability, robustness, and fault tolerance. DHTs distribute data across a number of peers, and the routing scheme from a particular P2P network is used to efficiently perform file lookups. P2P

networks have been widely adopted for content distribution because such solution does not require central servers to host the contents. In the traditional client-server architecture a request sent from a user to the server asking for a file is referred to as a file lookup in P2P computing. Each P2P network implements their own routing mechanism that works most efficiently in its architecture.

Most P2P networks face the problem of churn. Churn can be characterized as being the result of highly variable processes representing behaviour of users who are joining or leaving the system with or without cooperation from other users. In the worst scenario if a non-cooperative user leaves the system, then connections between its neighbours may require re-connections. Fluctuating number of users and changes of topology in P2P networks are considered to be more stochastically dynamic than in any other types of networks.

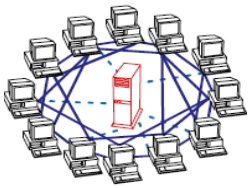
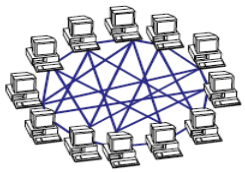
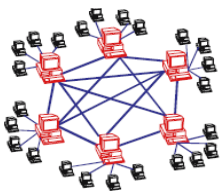
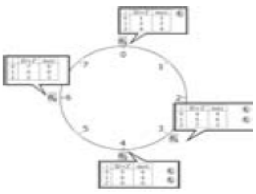
Unstructured P2P			Structured P2P
Centralized P2P	Pure P2P	Hybrid P2P	DHT-Based
Central management is required to provide file indexing and resolve file lookups Example: Napster, KaZZa [14].	Peers connect to each other without central control. Each peer shares the same amount of responsibility. File lookups are routed through flooding. Example: Gnutella 0.4	Peers are connected to high capacity peers, and rely on them to resolve file lookups. Examples: GIA	Peers are connected to each other with consistent hashing and adopt DHTs. Example: Chord, Kademlia
			

Table 2.1 Classification of structured and unstructured P2P networks [24]

Table 2.1 summarizes the classification of P2P networks. In centralized networks such as Napster and KaZZa peers are connected to central indexing servers and rely on them to resolve file lookups. An example of the Napster network architecture is presented in Table 2.1 under 'Centralized P2P'. Connection between peers is established after the indexing server replies to the source peer with the location of the target file. In pure P2P networks such as Gnutella peers are connected with each other and rely on each other to resolve file lookups, by routing their file lookups with the flooding method. An example is the Gnutella network architecture; see Table 2.1. In this network all peers are equally important and connections between peers are made without central control. In hybrid networks, such as GIA, peers are connected to high capacity peers and route file lookups to them because they know the location of more peers. An example of the GIA network architecture with leaf peers connected to high capacity peers, see Table 2.1. Structured P2P networks, such as Chord and Kademlia, adopt consistent hashing to evenly spreading peers out, so no peer is overwhelmed. The structured networks use their own routing methods to route file lookups. Chord uses scalable key location, where file lookups get routed to the peer nearest to the file match, and that particular peer will resolve the lookup. Kademlia uses the XOR mechanism to calculate the distance to other peers and routes file lookups to the closer peers in its routing table. An advantage of adopting DHTs is that any particular peer only needs to know a subset of peers in the network and still be able to route file lookups across the whole network. An example of the Chord network architecture is presented in Table 2.1 under 'DHT-based', with peers connected in a ring structure.

2.1 Structured network - Chord

Chord is one of the original structured P2P networks. It implements DHTs to perform resource discovery and management.

Chord possesses four features to maintain an efficient P2P communication:

- Load balance – Chord utilizes consistent hashing to achieve a balance in its network, spreading keys evenly across nodes. This will avoid any network

bottlenecks and the scenario where particular nodes get overwhelmed with resolving lookup queries [46].

- Decentralization – Chord maintains a topology in which each node is equally important. This improves the robustness of the network.
- Scalability –File look-ups take O(logN) time to complete, where N is the size of the network [46].
- Availability – Chord nodes make periodic updates to their routing tables reflecting on changes to the network such as newly joined nodes and disconnected nodes. This process is effective to reduce the effects of churn.

One of the key mechanisms Chord adopted is consistent hashing. The hash a node's IP address will output a key and it will be associated to the node. Consistent hashing tends to balance load where all nodes will receive roughly the same amount of keys, and mapping those keys to nodes that are responsible. In a large network it will be inefficient for a node to try to store the location of all other nodes in its routing table. Chord maintains scalability of consistent hashing by requiring a node to know the location of a few other nodes and because the information is distributed, a node resolves the hash function by communicating to other nodes.

When a N-node network is in a steady state, each node will only maintain around O(log N) other nodes, and resolve look-ups with O(log N) messages to other nodes. The churn behaviour of nodes joining and leaving the network is maintained in the routing information each node keeps. The network performance can be maintained at a satisfactory level as long as each node knows the location of O(log N) nodes in the network. In the case of simultaneous failures of nodes in the same subset of routing information, performance will degrade gracefully. As long as there is one correct copy of the routing information, look-ups will still be able to reach this destination with trade off in delay.

2.1.1 Initialization

Chord uses the Secure Hash Algorithm (SHA-1) as its base hash function to assign each node and key an *m*-bit identifier. The SHA-1 is able to produce a 160-bit digest with a maximum node address length of $2^{64} - 1$ bits [15]. The node identifier is derived from hashing the node's IP address, and the key identifier from hashing the key. The identifier length *m* must be large enough to ensure that two nodes or keys hashing to the same identifier is unlikely. Identifiers are placed on an identifier circle (known the Chord ring) modulo 2^m, and key *k* assigned to the first node whose identifier is equal to or follows *k* in the identifier circle, where the node is also known as the successor node of key *k*.

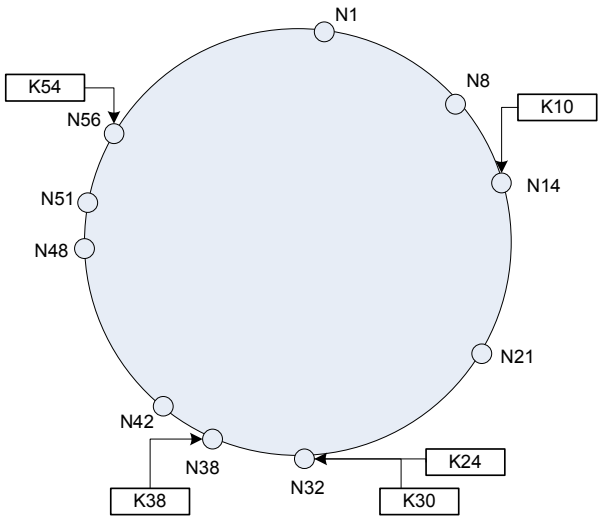


Figure 2.1 An example of a Chord network with 10 nodes and 5 keys.

Fig.2.1 depicts a Chord ring with m = 6, ten nodes and five keys. The successor of identifier 10 is node 14, so key 10 will be located at node 14. Keys 24 and 30 will be stored at node 32, key 38 at node 38, and key 54 at node 56.

To maintain consistent hashing under the effects of churn, keys are copied and given to a newly joined node from its new successor. For example, as node *n* (*n* = 1, 2, 3…N) joins the network, the set of keys previously assigned to *n*'s successor now become assigned to *n*. When *n* leaves the network, its set of keys is reassigned to *n*'s successor.

2.1.2 Simple key location

A simple search method is illustrated in Figure 2.2, where a search is performed by asking every node in the network to resolve the lookup. One can distinguish the following stages of this process.

The pseudo code of simple key location is shown below:

n.**find_successor**(*id*)

if(*id* ∈ (*n*, *successor*])

return *successor*;

else

return *successor*.*find_successor*(*id*);

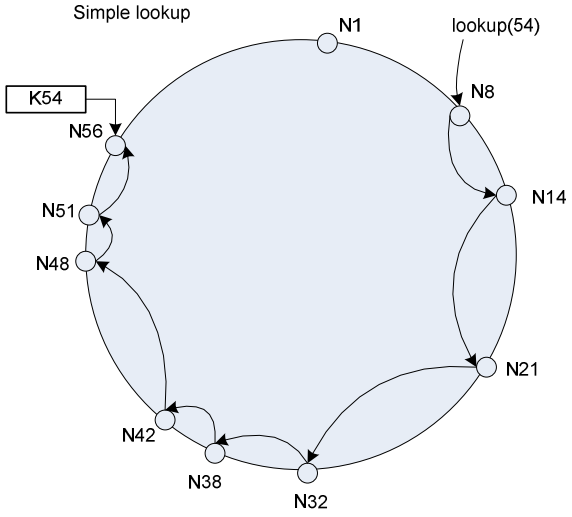


Figure 2.2. An example of Chord performing a simple lookup where node 8 is initiating a simple lookup for key 54 using simple lookup

In Fig 2.2, node 8 initiates a lookup for key 54. Node 8 will forward the lookup message to node 14 and if node 14 does not have the key, it will forward it on to the next node it knows, node 21 and so on.

2.1.3 File lookups - Scalable key location

The simple lookup method presented in 2.1.2 follows a linear path through all the connected nodes. This method is not efficient in scaling with larger networks, and the search time will take O(N) time. Presented in this section is a method called scalable key lookup.

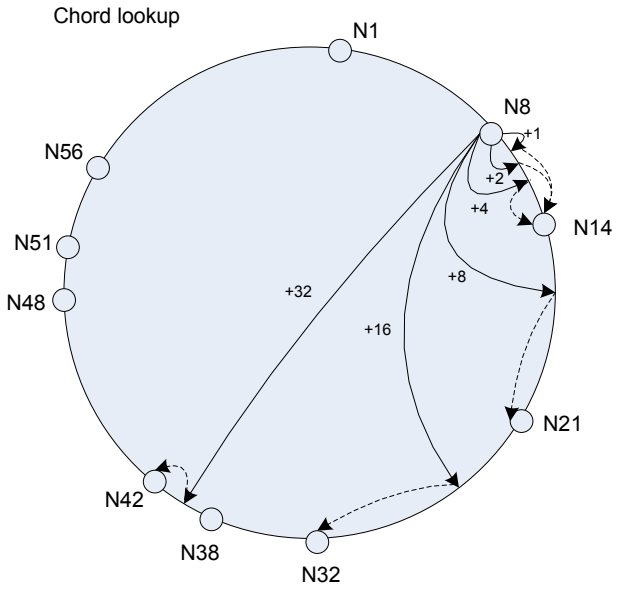
In a Chord ring with n peers, let m be the number of bits in the key/node identifiers. Each node n will maintain a finger table with up to m successors. The ith entry in the table at node n contains the identity of the first node s that succeeds n by at least 2^{i-1} on the identifier circle. For example, $s = successor\ (n + 2^{i-1})$, where $1 \leq i \leq m$. A finger table entry is composed of the Chord identifier and the IP address of the relevant node. In Fig. 2.3, the first entry in node 8's finger table points to node 14, because node 14 is the first node that succeeds $(8 + 2^0) \bmod 2^6 = 9$. The last entry in the finger table points to node 42, because node 42 is the first node that succeeds $(8 + 2^5) \bmod 2^6 = 40$.

The pseudo code of scalable key location is shown below:

```
n.find_successor(id)
if(id ∈ (n, successor])
        return successor;
else
   n' = closest_preceding_node(id);
   return n'.find_successor(id);

//search the finger table for the highest predecessor of id
n.closest_preceding_node(id)
for i = m downto 1
   if(finger[i] ∈ (n, id))
      return finger[i];
return n;
```

This implementation defines two important characteristics of Chord; first, each node will only be required to store the routing information of a small proportion of nodes in the entire network, and nodes that are closer to each other are better understood. Secondly, any particular node's finger table is not likely to store adequate information to directly locate the successor of a random key *k*. Fig 2.4 has an example of node 42 asking other nodes in its finger table that the source node, node 8, does not know about to help resolve the lookup.



Node 8 Finger table	
N8+1	N14
N8+2	N14
N8+4	N14
N8+8	N21
N8+16	N32
N8+32	N42

Figure 2.3. Node 8's finger table in a Chord ring network

Node 8's knowledge of other nodes in the network is displayed in Fig 2.3. Lookups will be sent to the closest node it knows, which are node 14, 21, and 32.

An example of Chord lookup is presented in Figure 2.4; node 8 initiates *find_successor* for key 54, and queries will be forwarded to relevant nodes in the finger table. Node 42 is the last entry in the node 8's table, so node 42 will resolve the query from its finger table, querying node 51. In node 51's finger table, it recognized that node 56 succeeds key 54, so it would return node 56 back to node 8 on the reverse path.

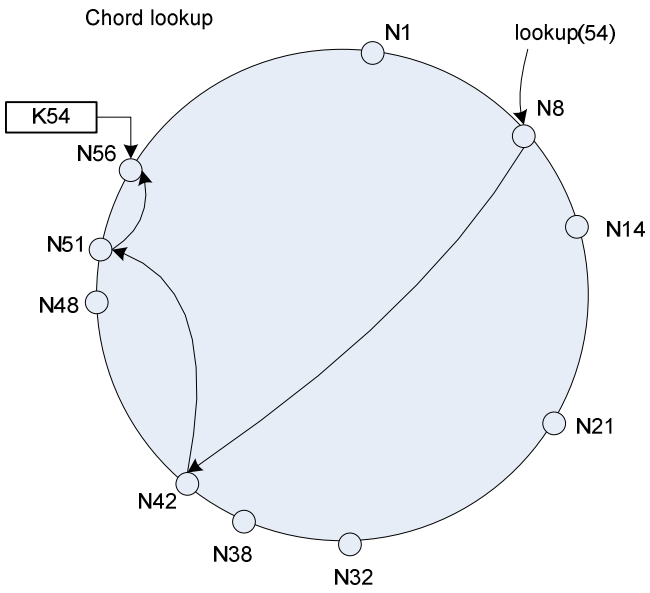


Figure 2.4. An example of a file lookup process where node 8 initiates lookup for key 54

2.1.4 Dynamic operations and failures

This section explains how Chord adapts when new nodes join the network, and voluntarily or involuntarily leave the network. Chord periodically executes stabilization mechanism to update each node's finger table and successor pointers; lookup requests can be resolved correctly.

Figure 2.5 presents a node join scenario. Node 26 joins the network between node 21 and 32. The consecutive operations are shown in the subfigures.
(a) Node 21 is currently pointing to node 32 as its next successor.
(b) Node 26 locates node 21's successor, node 32, and points to it as its own successor.
(c) Node 26 copies keys that are less than 26 from node 32.
(d) The stabilization process updates the finger table and pointers between node 21, 26 and 32.

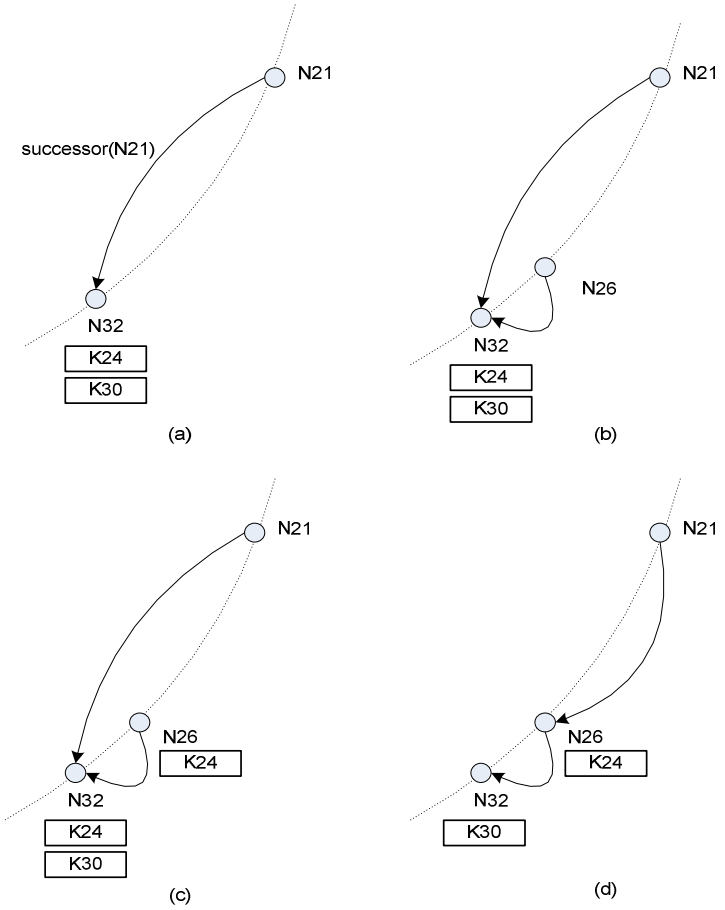


Figure 2.5. An example on the process of how node 26 joins the Chord network

Nodes can sometimes join the network on lookups and before stabilization has finished. Any of three scenarios presented below can happen:

Scenario A: Finger table entries are mostly up to date, and the lookup locates the correct successor in O(log N) steps. Performance wise, once stabilization finishes, lookup time will be maintained at O(log N) time.

Scenario B: Successor points are correct but finger table entries are incorrect. The lookup will still be successful, but performance might be delayed. In this scenario, lookup queries can still be passed on around the network because it does not depend on exactly which nodes the entries point to. A node knows the location of other nodes that are very distant, so lookup performance is not affected as much.

Scenario C: Incorrect successor pointers or keys might not be copied to the new node yet. The lookup might fail, and the application may choose to reinitiate the lookup process again. If the new node happens to be between the target's predecessor and the target, it will depend on the new node to stabilize itself with updated finger table to resolve lookups.

2.1.5 Failures and replication

For Chord to be efficient, nodes need to have a correct finger table, but if multiple nodes fail simultaneously then the lookup might fail. For example, in Fig 2.3, if nodes 14, 21, and 38 fail simultaneously, node 8 will not know that node 38 has become its successor because it did not have a pointer to node 38 in its finger table. At the same time, if node 8 initiates a lookup for key 30, node 42 will respond as it had become the first entry in node 8's finger table. However, the correct node to respond should be node 38.

To improve the robustness of a Chord network, one can consider increasing the size of the finger table in each node. It will be very unlikely for multiple successors to fail simultaneously in any particular finger table. However, a balance between correctness of file lookups and delays should be considered to achieve desired performance.

2.1.6 Summary

Chord is designed to provide co-operative file sharing, time-shared file sharing, and distributed indices for document and service discovery in a decentralized architecture. It is considered to have a good scalability. In the steady state, in a network of size N, each node will only need to store the routing information of O(log N) other nodes, and to resolve lookups in O(log N) messages to other nodes. It also adapts to a dynamic situation where there are constant disruptions of nodes joining and leaving.

2.2 Structured network – Kademlia

Kademlia, like Chord, is a structured P2P network. It has a similar structure of implementing DHTs, but it has new features to improve efficiency of file lookups. One of the differences between the Chord and Kademlia is the use of the XOR mechanism in Kademlia to calculate distance between two nodes. For example, in case of node x and y with their 160-bit identifiers, Kademlia can use XOR to assess the distance as $d(x,y)=x \oplus y$.

The main features of Kademlia include:

- Bandwidth preservation – network management messages can piggy back on key lookups.
- Asynchronous queries – use of parallel and asynchronous queries to avoid timeout delays from disconnected peers.
- Efficient routing – routing queries through low-latency paths.
- Distance calculation – use of XOR metric. This is a symmetric metric, so each node receives approximately the same number of lookup queries as the number of neighbours it has.

Like Chord, Kademlia's XOR is also uni-directional, where lookups on the same key converge on the same path, no matter where the lookups were initiated. Kademlia uses the XOR metric to calculate the distance between two nodes. XOR returns zero if two bits are equal and one if two bits are different.

XOR properties

- $d(x, x) = 0$
- $d(x, y) > 0$, if $x \neq y$
- $\forall x , y : d(x, y) = d(y, x)$
- Triangle property
  - $d(x, y) + d(y, z) \geq d(x, z)$
  - $d(x, z) = d(x, y) \oplus d(y, z)$
  - $\forall a \geq 0, b \geq 0 : a + b \geq a \oplus b$

Other structured P2P systems, such as Chord that does not implement the XOR mechanism, require additional algorithms for locating targets that share the same prefix but differ in the next *b*-bit digit. This algorithm costs more time, and requires secondary routing tables of size $O(2^b)$ and main routing tables of size $O(2^b \log_2{}^b N)$. Kademlia can be improved with a base other than 2, where buckets approach target *b* bits per hop. The maximum amount of buckets are expected to be less than $(2^b - 1) \log_2{}^b N$. The default value of *b* is set to 5.

2.2.1 Initialization

During initialization, each Kademlia node is assigned a globally unique identifier (GUID) as its node ID. Each node stores key and value pairs. The value parameter stores file hashes or keywords. To ensure the uniform distribution of keys, a hash (160-bit SHA1 digest) of value is derived. The initialization procedure of Kademlia is very similar to Chord. Every Kademlia node stores a list of (IP address, UDP port, Node ID) triples for nodes of distance between 2^i and 2^{i+1} from itself. The lists are also known as *k*–buckets.

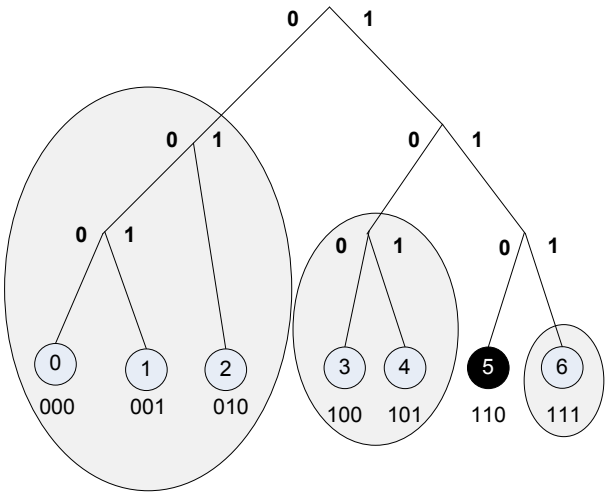


Figure 2.6 An example of a Kademlia network with 7 nodes and 3 buckets.

In Figure 2.6, the network size is 2^3, seven nodes exist with a maximum of eight nodes. We are looking at the network from node5's perspective. Node 5 (110) has three *k*-

buckets at the moment. The first bucket contains node zero (000), one (001) and two (010). The second bucket contains node three (100) and four (101). The third bucket contains node six (111). We can see that node six is the closest node to node five, with a distance being 1.

Distance between node six, one, five, and seven.

node one	001	node four	101	node six	111
node five	110	node five	110	node five	110
XOR	111	XOR	011	XOR	001
Distance	7	Distance	3	Distance	1

2.2.2 File Lookups

Similar to Chord, Kademlia uses an algorithm to search closest nodes to the target. The algorithm is recursive and asynchronous. Each Kademlia client can select α nodes and simultaneously initiate lookups. α is a pre-determined value typically to be three set by Kademlia developers. In the recursive process, the source node sends out RPCs to new nodes found by previous RPCs. The recursive RPCs can occur before all previous RPCs have returned. During a lookup, nodes that have not responded within the time limit will not be considered for future lookups until they respond. After a cycle of node lookups, if the source node has not seen results that are new and closer to the target ID, it will initiate another set of RPCs to closest nodes that have not been queried. When the intermediate recipient nodes receive the request, they will look through their bucket entries, and reply to the source node with the closest nodes they know in reaching the target node. The source node will update its bucket entries and resend the requests to the previously known closest nodes for more answers, and this process will be iterated again and again because each node has more knowledge of its surrounding nodes than nodes that are further away. The iteration stops when no new closest nodes can be found, and the current closest nodes listed in the bucket are closest to the target key.

When α = 1, the search process is similar to Chord in bandwidth consumption and latency from identifying non responsive nodes, but Kademlia is more efficient on routing because it can choose which node to route the request to.

The FIND_VALUE RPC is similar to the FIND_NODE RPC. The goal is to find the closest IDs to the desired key. FIND_VALUE takes in a (key, value) pair as its argument to initiate a search. In FIND_VALUE, the search process stops when a node responds back to the source node with the value. To improve the efficiency of future searches, the value retrieved from previous FIND_VALUE RPC will be cached to the closest key that did not return the value. Kademlia has a unidirectional topology, and searches on the same key are likely to locate the cached value before reaching the closest node.

Remote Procedure Calls (RPCs)

PING	used to identify if a node is still alive
STORE	used to store a (key, value) pair in a node
FIND_NODE	given a 160-bit ID to lookup, the recipient of this RPC returns the (IP address, UDP port, Node ID) triples to the closest node it knows where a match might be found. The recipient will return *k* nodes in its own buckets that are closest to the requested ID.
FIND_VALUE	similar to the FIND_NODE RPC, returns triples from the recipient node, but if the recipient has received a STORE RPC for the key, it will return the stored value.

Looking at Figure 2.7, one can see that the size of the bucket reduces down to two instead of three. The left-most bucket now can only contain node one (001) and node two (010). Node 5 will now have to rely on this bucket of nodes to resolve queries to node zero.
Joining nodes need to go through the bootstrapping phase to join a network: new nodes must first make a connection to a node that is already participating in the network. For example, a new node *x* inserts (IP address, port) details of node *y* into its own *k*-bucket, and then performs a lookup to *y* on itself. This process will add a new entry in other

nodes' *k*-buckets with node *x* ID, and also populate *x*'s *k*-buckets with the nodes between itself and the bootstrap node.

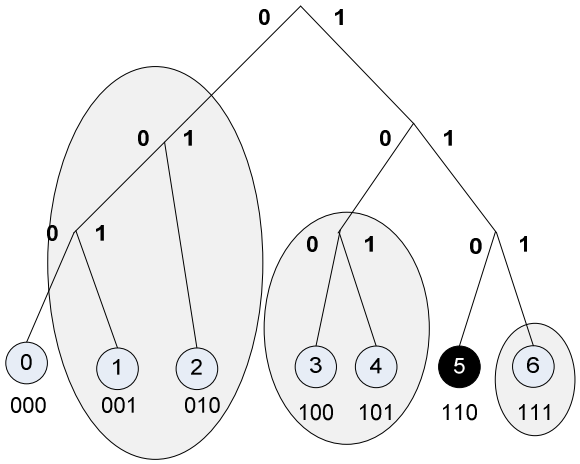


Figure 2.7 Example of the the same network as in Figure 2.1, but with the maximum bucket size reduced from 3 to 2 nodes.

2.2.3 Stabilization

Entries in each bucket are sorted from most recently visited nodes at the bottom to previously known nodes at the top of the list. The value of *k* is specifically chosen so that the group of chosen nodes is unlikely to be disconnected within an hour of each other. Kademlia nodes update their *k*-buckets when they receive any request or reply messages from other nodes. If a node ID from the received messages matches an entry in the *k*-bucket, that particular entry gets moved to the bottom of the list. If it is an ID that cannot be recognized and the bucket is not full yet, the new ID will be inserted at the bottom of the list. However, if the bucket is full, the node will first ping the entry at the top of the bucket where it has not been heard for a while and then make a decision. If there is no response from the pinged node, then it will be removed, and the new ID is inserted at the bottom of the bucket. On the other hand, if the node responds, its entry will be moved to the bottom of the bucket, and the new ID will be ignored. Kademlia adopts the method of removing non-responsive nodes and always keeping live nodes. This implementation was backed up by Kademlia's designers following their analysis of Gnutella trace files. The

assumption is that the longer a node has stayed alive, the more likely it is to remain alive in another hour.

2.2.4 Summary

Kademlia was designed to maintain good performance under churn and to reduce bandwidth consumption during routing. It implements the XOR distance metric to speed up its file lookup process. Kademlia also applies mechanisms that consider client's available bandwidth, for reducing latency related with hop selection and delay-free fault recovery.

2.3 Unstructured network - GIA

GIA is an unstructured P2P network. The developers of GIA tried to create a robust Gnutella-like system while correcting its flaws and improving scalability.

The main components of GIA are:

- Dynamic topology adaptation – a protocol that connects nodes to high capacity nodes. The protocol also periodically verifies that high capacity nodes have sufficient resources to handle file requests.
- Active flow control – this mechanism prevents overloading of nodes with file requests. This protocol periodically updates the status of high capacity nodes, and assigns tokens pending on available resources.
- One-hop replication – each peer node maintains a list of contents their neighbours have. With the dynamic topology adaptation, each peer is connected to a high capacity node, the one-hop replication ensures that high capacity nodes are able to resolve a large amount of queries.
- Biased random walks – this search method ensures that queries are only sent to high capacity nodes that are more capable of resolving queries.

Unstructured P2P networks, such as Gnutella, face problems under high lookup rate and when they operate with a large number of peers. GIA is built to maintain performance under the stress of high query rates. The second goal is to resolve scalability issues when networks grow larger, and avoid overloading nodes by being aware of their resource constraints.

Gnutella uses a flooding-based search method to perform file lookups. The source node initiates its file request to its neighbours, and the request gets propagated to other nodes until it reaches certain constraints set by the source node. This type of search method achieves high success rate with the trade off of bandwidth consumption. To overcome the problem, other work has suggested using random walks instead of flooding.
Random walk forwards file requests to randomly chosen neighbours until sufficient results are found. However, it still poses two problems:

- The random walk search method will blindly forward file requests to other nodes with no consideration on how likely the target node will be able to resolve the request.
- The random walk search method will not take network status into consideration, and requests might get forwarded to an already overloaded node.

To overcome these problems, GIA implements biased random walks. File requests will only be forwarded to high capacity nodes. The capacity of a node is evaluated by its processing power, disk latencies, and available bandwidth. High capacity nodes with more resources are likely to have more neighbours, and are more aware of where files are.

2.3.1 Initialization

GIA clients maintain a list of other GIA clients, a host cache, which stores the high capacity nodes. The cache is updated by contacting web-based host caches and exchanging host information with neighbours. The web-hosted cache will periodically update the list of high capacity peers and remove disconnected peers.

2.3.2 Topology Adaptation Process

Topology adaptation process makes sure that the right nodes are selected to be the high capacity nodes, and lower capacity nodes are closely connected to them. To accomplish this, nodes compute an indication value also known as level of satisfaction. Its value varies between 0 and 1, and describes how satisfied a node is with its set of neighbours. A value of 0 indicates that the node is dissatisfied and 1 means that the node is satisfied with its current set of neighbours. Topology adapts by searching for new neighbours, trying to achieve a better satisfaction level. At start up, each node will have a value of 0 and as it finds more neighbours, the satisfaction level increases.

The pseudo code describing the topology adapatation process is presented below:
Let *max_ nbrs* mean the maximum number of neighbours.
Adding new node, node Y
if $num\_nbrs_X + 1 \leq max\_nbrs$ **then** //there is room
    ACCEPT *Y* ; return

//need to drop a current neighbour to make room
subset←$i \ \forall \ i \in nbrsX$ such that $Ci \leq C_Y$
if no such neighbors exist **then**
    REJECT *Y* ; return
candidate *Z* ←highest-degree neighbor from *subset*

if ($C_Y > max(Ci \forall \ i \in nbrs_X)$) //node Y has more capacity
 or ($num\ nbrs_Z > num\ nbrs_Y + H$) //node Y has fewer neighbours
  then
    DROP *Z*; ACCEPT *Y*
else
    REJECT *Y*

If node *X* wishes to add a new neighbour to improve its satisfaction level, it randomly selects a few nodes in its host cache that are alive, and not already a neighbour. Node *X*

will choose to connect to nodes with a capacity greater to its own, and if no suitable options exist, it will select a random node. Before a connection is established, both the source node (node *X*) and target node (node *Y*) will need to agree to accept each other as a neighbour, pending on available capacities and number of current neighbours. Node *Y* might need to drop one of its neighbours to make room for node *X*.

Node *X* will first check if its number of neighbours is within the limit, and if so, node *Y* will automatically be accepted. If not, node *X* will need to determine which of its neighbour to remove to make room for the new node. If new node *Y* has more capacity than any of *X*'s neighbours, *Y* will always take the place of another node. On the other hand, if *Y* has the same or less capacity than any of *X*'s neighbours, *X* will choose to drop one of its neighbour with the highest degree, *Z* for example, to accommodate *Y*. Node *Z* will be less affected when dropped, because it is the node with the highest degree in *X*'s list. This design will protect any current poorly-connected node from losing its connection with high capacity nodes.

2.3.3 Flow Control

The flow control mechanism was designed to prevent any node from being overloaded. This mechanism limits each node's right to forward requests. GIA implements a token-based system, nodes are allowed to forward requests based on the number of their tokens they collect. GIA nodes periodically assign tokens to their neighbours. The rate each node dispenses tokens are based on their speed of resolving and responding queries[1]. In the event where a node is receiving more queries than it is forwarding, it will lower the rate of dispensing tokens to its neighbours.

To ensure that the majority of queries are forwarded to higher capacity nodes, GIA clients assign tokens pending on the node's capacity; the more capacity a node has the more

[1] The GIA flow control mechanism in OverSim periodically issues tokens without assessing the source peer's available resources.

tokens it distributes to its neighbours. Unused tokens will be discarded, and the previously reserved capacity will be redistributed between its neighbours. Tokens can be sent in separate control messages or by piggy-backing on other messages.

2.3.4 One-hop Replication

Each GIA client maintains a list of contents from each neighbour that it is connected to. GIA clients share their list of contents with their neighbours when a connection is established, and periodically update the list. This will improve the success rate in resolving a query; the target node will also be able to compare the query against its neighbours' list of contents. To assure the validity of the entries in the list, whenever a connection to a particular neighbour is lost, its relevant entries are also removed.

2.3.5 File Lookups

Topology adaptation and one-hop replication can ensure that high capacity nodes do have the resources to resolve queries, and the knowledge of the locations of more contents. GIA clients implement biased random walk, where queries are directed to the high capacity nodes, rather than to a random selection. Clients will need to wait for the flow control tokens to send out queries. The time to live (TTL) parameter is used to limit the duration of the biased random walk queries, and the book-keeping method is used to avoid redundant paths. In the book-keeping process, nodes assign their query a unique GUID. The source node remembers which neighbour it forwarded its particular GUID query, and if the same query returns it is forwarded on to a different neighbour. When the query has been forwarded to all neighbours, the book-keeping state is reset, and all neighbours are open for selection to forward queries to again. Queries have a MAX_RESPONSES parameter, indicating the maximum number matching result the queries should search for. If a match is found during a search, the parameter MAX_RESONSES decrements in the query, and the query is discarded when MAX_RESPONSES reaches zero.

2.3.6 Summary

GIA uses an improved architecture, which is an improved version of the original Gnutella design. It has added functionalities such as flow control, dynamic topology adaptation, one-hop replication, and a new search protocol. The new design will preserve bandwidth, maintain efficiency and success rate of resolving queries, and increase scalability and robustness.

Chapter 3

Experiment Design

In this chapter we discuss the methodology used to evaluate the performance of file lookup mechanisms in P2P networks and issues that may affect the accuracy of our results.

3.1 Methodologies

P2P algorithms are often studied by using crawlers, emulators or simulators [35]. In such crawler-based experiment, crawlers would be specifically built peer nodes which collect data from other nodes by visiting those nodes. By deploying multiple crawlers, a large proportion of a P2P network can be monitored. One of the drawbacks of this approach would be the fact that knowledge of a given P2P network would be limited to regions where the crawlers have been deployed. The other disadvantage of using crawlers is that users will have to be aware of how much overhead is produced while monitoring the actual P2P network experiment and will need to take this into account when interpreting the results. A more effective scenario would be to use emulations. Multiple emulators can run within a workstation to imitate peer nodes and form a complete P2P network. The downside of emulation is that although it does not affect real peers, it has a low efficiency because messages sent between nodes are processed just like in a real scenario, thus the processes can be slow. The third approach, based on simulations, often use simplifying assumptions in the network access and/or Internet level of the TCP/IP model, to increase the speed of simulated overlay protocols. This approach is more popular as it allows more algorithm studies. However, without inputs in the network access level and Internet level, results can be not satisfactorily accurate. On the other hand, with abstracted TCP/IP simulation models one can afford to analyze more complex overlay systems in a reasonable time. In simulations, users are able to control the level of abstraction assumed.

The level of abstraction can be categorized into packet-level, overlay model or stream approach.

Model	**Abstraction level**	**Accuracy**	**Computation complexities**
Packet level	Low	High	High
Overlay model	Medium	Medium	Medium
Stream approach	High	Low	Low

Table 3.1 Comparison of different abstraction level in simulations

From Table 3.1, we can see that simulations with lower abstraction level can produce more accurate results but require intensive computation. Thus, the size of networks simulated with such details is typically smaller. Simulations with higher abstraction level assumed can afford to be scaled up to larger networks, but the accuracy of results needs to be reduced. The balance between selecting the most suitable simulation approach depends on the goal one tries to achieve through simulation.

3.2 Survey of P2P Simulators

Several criteria for assessing P2P simulators were suggested in [39]. We have adopted and modified it to identify the suitable simulator for our research project, by taking into account:

Simulation Architecture –Main features of design and functionalities of simulators, and how they are implemented. An example of this would be to distinguish if the churn can be modelled. To better capture the behaviour of P2P users, simulated churn should be able to input parameters into probability distributions such as Weibull and Pareto. The extendibility of the simulator should be also assessed, which includes easiness of implementing other P2P protocols and collecting more simulation data.

Usability – Here, the ease of learning and using the simulator is considered, including its documentation. Each simulator should include a common API. A well defined API would allow the simulator and protocol code to be easily understood and re-implemented. The other aspect to consider is how easily and precisely a simulation can be set up, executed, and manipulated.

Scalability – Most P2P protocols are designed to effectively deal with increasing number of peers. The assessment here would be to examine how large a simulated network can be stabilized. Session level simulations should be able to support more than 10,000 nodes, and packet level simulations should be able to support more than 1,000 nodes.

Statistics – The ability to collect output data and calculate basic statistics is an important aspect of simulators, which is often over-looked. The output data need to be readable and easy to modify for statistical analysis later on, although preferably a simulator should have a possibility of on-line analysis of output results.

Underlying network – Simulators take different approaches in modelling underlying networks. They can simulate packets in detail, or completely abstracting anything under the overlay layer. This feature includes the properties of the network layer which can be simulated, modelling of background traffic and link latencies etc. Simulators that can model more network properties are expected to be able to produce more accurate results.

In our survey we have studied a few simulators including: Network Simulator 2 (NS2) [3] and Network Simulator 3 (NS3) [7], PeerSim [4], OverSim [1], Neurogrid [5], PlanetSim[13] and P2PSim [6]. As the result of our survey we have chosen Oversim to be used in our studies. It is a discrete event simulator which allows users to define a specific event and terminates the simulation based on the occurrence of that event. This simulator also has a clear separation between the application layer, protocol overlay and network underlays. The module design also helps to ease customization of experiments and improves extendibility of functionalities. Oversim is designed to operate on top of another

network simulator, OMNeT++ [2], which allows more accurate simulations if users can define more underlying network properties.

Oversim is actively maintained by its developers, and users can get full documentation with simulator APIs online, or further help through email correspondence with the developers. This simulator also adopts a GUI similar to that used in OMNeT ++, assists users to customize experiments, and visualizes active simulations. Oversim also provides an extensive data collection in its trace files. One of the most important reasons for choosing Oversim is that it is equipped with different churn models, based on such different probability distributions of peers' lifetime as Weibull, Pareto and log-normal. These functionalities and advantages of Oversim have influenced our decision of using it as the simulation tool in our research project.

3.3 Simulated Network size

It is common for a typical real world P2P file sharing network to grow up to tens of thousands, or even millions of peers at any particular time. In our simulation studies, we wanted to capture as much characteristics as possible from the real world, for adding credibility to our results. One of the issues of our simulations was the selection of the size of simulated networks as limited computing power of current computer installation limits our ability to simulate larger networks. Unfortunately, using typical computing system of today, it becomes unfeasible to perform simulation if the simulated network size grows to the magnitude seen in the real world. For example, research projects reported in [31], [33], [36] and [50], studied network underlays for P2P systems of size ranging between hundreds to 2,400 nodes. On the other hand, in [49] where network underlay was not considered, the size of simulated P2P network was able to grow up to 10,000 nodes. To add accuracy in our research studies, we have opted to simulate the network underlay. Within OverSim, it is possible to use the OMNeT++ network simulator to simulate the TCP/IP stack. The network underlay in OMNeT++ provides simulation of a number of mechanisms of network underlay, such as network backbone delays and peer bandwidth requirements which add more realism to our simulation study.

With the computing technology and the increased complexity from adopting the network underlay, the maximum size of our basic simulated network has been set to 100 nodes. In the study of scalabilities, the size of the network has been increased to 1,000 nodes.

3.4 Churn Modeling

One of the major problems suffered by P2P networks is churn. Churn is a term referring to random disruptions of connections between peers. In [50], the churn has been categorized into environmental and characteristic factors. In the environmental factors the parameters concerned are number of active peers and their joining frequency. In characteristic factors, the parameters concerned are the retransmission times of query failures.

One of the most discussed problems is the impact of peers joining and leaving the network. Joining peers are considered to have weaker influence on a given P2P system, because they generally result in short term failures, and scheduled routing updates can resolve the problem. Disconnecting peers are considered to impair the network more as they may lead to loss of overlay structure or loss of data availability in the overlay [23]. Disconnecting peers can be categorized by graceful failures or unexpected failures. Peers that disconnect gracefully will notify their neighbours to restructure the topology before they leave the network. In the case of unexpected failures, neighbours of the disconnected peer would not be notified before the failure happened, and it would not be until the next scheduled routing table updates or file lookup arrived at the location of the disconnected peer that the network would notice the change. Churn affects many aspects of P2P computing, including lookup efficiency and network stabilization. It is crucial to capture these characteristics to increase the accuracy of our performance evaluation studies.
There have been many debates over which churn model is more accurate in modeling the real P2P users' behavior. Various studies have argued that Weibull ([20], [40], [48]), Pareto ([30], [51]) or Poisson ([22], [36], [42]) probability distributions are the best models for describing the P2P users' behaviour. In a Poisson churn model, the "join" and "leave" processes are considered to be streams of events which follow exponential

probability distributions and the joint process of number of these events is modelled by a Poisson probabilistic distribution with rate λ. As pointed out in [22], simulating a number of peers joining a network with a given time interval, which is independent of the current size of the network, will create an additional problem. In a small network with few hundreds of peers and λ=100, the simulation results would be greatly influenced. On the other hand, in a large network with tens thousands of peers and λ=100, the simulation results would not likely to be influenced. In a Weibull- or Pareto-based churn model, distributions are generated by assuming that the durations of peer up times and down times follow these distributions. Some results indicate that durations of peer sessions in a typical P2P file sharing network follow a heavy tailed distribution [30]. For this reason, we have decided to base our models of churn on Pareto distributions.

This model is based on the results of [51]. The authors have proposed a Pareto distribution churn model to capture the behavior of a single user. A users' behaviour is based on the specified assumptions about peer arrival, departure and selection of neighbours. Namely, it is assumed that the P2P network has a total of N peers, and each peer i can either be active (connected to the network) at time t or sleeping (disconnected from the network). Such a dynamic behaviour is modelled by an alternating renewal process $\{Z_i(t)\}$ for each peer i

$$z_i(t) = \begin{cases} 1, & \text{if user } i \text{ is alive at time t,} \\ 0, & \text{otherwise.} \end{cases}, \quad 1 \le i \le N \tag{1}$$

Figure 3.1 illustrates a possible dynamic behavior of an individual user (peer). The subscript c stands for the cycle number of peer i. ON (active) and OFF (sleeping) periods are represented by random variables $A_{i,c} > 0$ and $S_{i,c} > 0$, respectively. The residual variable R_i, which is the duration of peer i's remaining ON (active) period from time instant t is also shown.

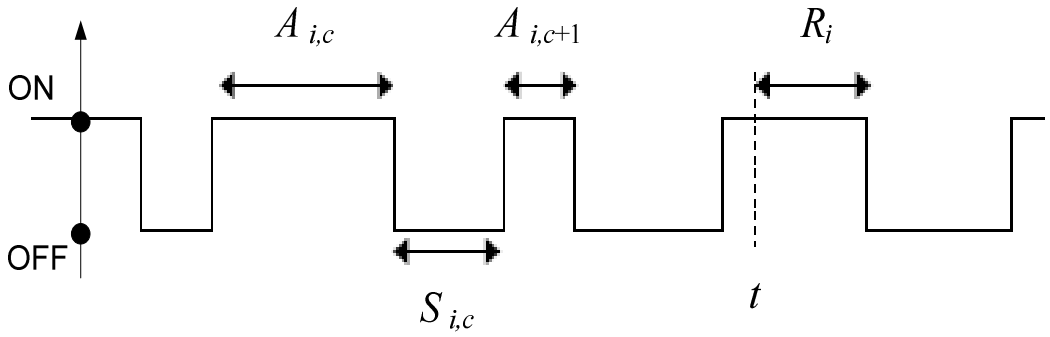


Figure 3.1 Churn model representing the ON/OFF behaviour $Z_i(t)$ of peer i.

The model assumes that peers do not synchronize their arrival or departure times and have uncorrelated lifetime characteristics. It is also assumed that it will be uncommon for peers to be present in the system with multiple identities, or in other words, we assume that peers with such multiple identities do not have a significant impact on the dynamics of the network. It is also assumed that all $\{Z_i(t)\}$ have their ON durations $\{ A_{i,c} \}_{c=1}^{\infty}$ governed by the same distribution, given by Eq. (2). A similar assumption applies to OFF durations $\{ S_{i,c} \}_{c=1}^{\infty}$ which are ruled by the distribution on Eq. (3) below.

The assumption that both ON and OFF durations are governed by a shifted Pareto distribution means that

$$F_{Ai}(t_a) = 1 - (1 + \frac{t_a}{\beta_{Ai}})^{-\alpha_{Ai}} \qquad \text{for } t_a > 0,\ \alpha_{Ai} > 1,\ \beta_{Ai} > 0\ , \qquad (2)$$

$$F_{Si}(t_s) = 1 - (1 + \frac{t_s}{\beta_{Si}})^{-\alpha_{Si}} \qquad \text{for } t_s > 0,\ \alpha_{Si} > 1,\ \beta_{Si} > 0\ . \qquad (3)$$

Further, since we are assuming that they are identical, $\alpha_{Ai} = \alpha_{Si} = \alpha_i$ and $\beta_{Ai} = \beta_{Si} = \beta_i$.

Let the average activity time be $E[A_i]$, and the average sleeping duration be $E[S_i]$. According to our assumptions, $E[A_i] = E[S_i] = \dfrac{\beta_i}{\alpha_i - 1}$.

The β_i value equals $\dfrac{1}{E[A_i](\alpha_i - 1)}$ for activity time, and $\dfrac{1}{E[S_i](\alpha_i - 1)}$ for sleeping time.

The peer's availability, i.e. probability that a given peer is in the network at a random instance t, t >> 0, is given by:

$$P_{Ai} = \lim_{t\to\infty} P(Z_i(t) = 1) = \frac{E[A_i]}{E[A_i] + E[S_i]} \quad ; \qquad (4)$$

for example, see [22], [43].

In simulations, during the network establishment phase, the network will populate itself to reach the maximum capacity, and each individual peer *i* will draw its active time from the distribution described by Eq. (2) and its sleeping time duration from Eq. (3). We can set the average peer session length in our simulations by adjusting the average active and sleeping parameters. Once the network is established, each individual peer *i* will stay connected in the network until its assigned lifetime, and disconnect from the network afterwards. The network will not add a new peer until the sleeping duration of peer *i* expires.

3.5 Routing method

P2P network messages can be routed with either iterative or recursive routing [25]. Peers using iterative routing can send file lookups to all of the nodes in their routing tables simultaneously. For example, as shown in Figure 3.2, P1 can send four file lookups to P2, P3, P4 and P5 simultaneously, and peers can reply straight back to P1. On the other hand, a peer using recursive routing can only send its file lookup to the first peer in its routing table and has to rely on the fact that the message will be forwarded. As Figure 3.2 shows, P1 can only send one file lookup at a time and relies on intermediate peers to forward its file lookup to the destination, and the same restriction applies to the destination peer when it replies back to P1.

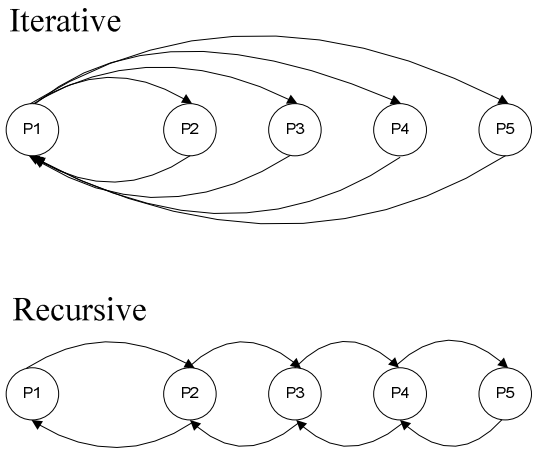


Figure 3.2 Iterative and Recursive routing

The main characteristics of the two routing methods are as follows:

- Recursive routing forwards file lookups to one peer at a time from its routing table, and if the peer with the lookup disconnects from the network, the lookup process terminates prematurely.
- Recursive lookups incur less latency compared to iterative lookups.

A more detailed discussion of these two types of routing can be found in [25].

3.6 Summary

In this chapter, we discussed the main properties of P2P systems which need to be taken into account if their performance models are constructed. The main properties are: (a) the size of population of peers, (b) probabilistic distributions governing churn properties, and (c) type of routing.

Chapter 4

Performance Evaluation

This chapter presents results of the performance evaluation study of Chord, Kademlia and GIA, and their lookup mechanisms in particular. The aim is to investigate the effects of churn on the performance of these P2P networks. The results were obtained by means of stochastic discrete-event simulation, using OverSim as a simulation tool. This tool was selected on the basis of the survey of simulators as reported in Chapter 3 and Appendix C.

4.1 Simulation setup

4.1.1 Assumptions

Assumption 1 – Behaviour of a user is described by alternative renewal process given in Eq. (1). Following our discussion in Chapter 3, it means that each user spends a time interval of random length in "activity" state A_i, followed by a time interval of random length in "sleeping" state S_i. In all cases, the lengths of these time intervals are governed by Pareto distributions given by Eq. (2) and Eq. (3). This assumes that all users operate in the same way, so they are governed by the same Pareto distribution.

Assumption 2 – A simulated P2P network operates at its full capacity from the beginning of simulation. It means that in the initial state of simulation all peers are active.

Assumption 3 – A P2P network has N homogeneous peers. Unless otherwise stated, we assume N=100. In section 4.2.3, N=1,000.

Their homogeneity means that $\alpha_i = \alpha$ and $\beta_i = \beta$ for all i, given in Eq. (2) and Eq. (3).

In all sections but section 4.2.1, routing is set to iterative routing. A more detailed study on the effects of iterative and recursive routing in file lookups in structured networks is presented in section 4.2.1.

There have been many arguments on the defining the average activity period that best reflects real P2P user behaviour. Studies from [31], [36], [42] have suggested average activity periods ranging from few minutes to one hour. In [32] we can see that the results become constant beyond an average activity period of one hour. We have assumed the average activity periods of a peer to be 900, 1800, 2700 and 3600 seconds, respectively. This allows us to capture the dynamics of simulated processes related with joining and leaving P2P networks by peers.

Our studies are intended for assessing the file lookup processes. We assume that once a match is found, the keyword from the file lookup is copied to the source node, and no content retrieval is performed. Following [31] and [32], we will consider the following performance metrics to represent the performance of file lookups processes:

- Hop count

  The hop count represents the distance between the source node who initiated the lookup and the destination node (the location of the searched value), measured by the number of intermediate nodes which needs to be traversed between the source and destination node.

- Bandwidth consumption

  The bandwidth consumption includes the amount of traffic generated by routing table updates and file lookups, measured by the number of messages generated during a fixed duration of time.

- Latency

  The latencies measures the duration of the stabilization processes from sustaining churn, as well as the duration of time needed for resolving file lookups from when it was initiated until it was responded to; measured in milliseconds.

- Delivery ratio in Chord and Kademlia

The delivery ratio measures ratio of successful deliveries of file lookups to the destination node over a given time interval.

- Satisfaction level in GIA

The satisfaction level represents the level at which a GIA peer is satisfied with its current set of neighbours in resolving its file lookups. The satisfaction level ranges between a value of 0 and 1, 0 being unsatisfied and 1 being satisfied. This value is calculated from the quality of responses the peer received from its set of neighbours in resolving its file lookups. For more details, see section 2.3.2.

4.1.2 Simulation Models

As described in Oversim manual, before a simulation can begin its users need to define their experiment in the *omnetpp.ini* script file. The default values of each P2P network, underlay network properties, and file lookup parameters are stored in the *default.ini* script file. If users choose to adopt the default values then they do not need to be declared in *omnetpp.ini.* On the other hand, if users choose to set up their own values, they have to declare them again in *omnetpp.ini.*

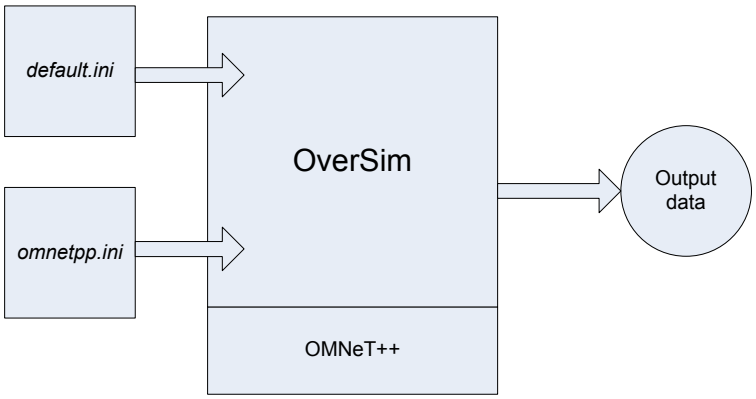


Figure 4.1 Simulation architecture

As shown in Figure 4.1, simulation parameters are defined in the *default.ini* and *omnetpp.ini* script files, which are the inputs to OverSim. OverSim is an overlay

simulator, and it makes use of OMNeT++ in the background to handle simulation details in the underlay. Output data is prepared by OverSim. As shown in Table 4.1, the *default.ini* file contains all the default values, and will remain unchanged during simulation unless otherwise overwritten in *omnetpp.ini*. The *omnetpp.ini* holds simulation scripts, where users can define their specific experiments. See Appendix B for the parameters used in our experiments.

default.ini	*omnetpp.ini*

• P2P network overlay
• Test application
• Network underlay
• Churn model
	• Simulation description
	  • Customized P2P network overlay
	  • Customized network underlay
	  • Customized test application
	  • Customized churn model

Table 4.1 Simulation configurations

4.1.3 Default simulation parameters

- Chord
  - Successor List Size: 8
  - Retries before a sucessor is considered failed: 2
  - Successor stabilization interval: 20 seconds
  - Check predecessor delay: 5 seconds
  - Join delay (delay between join retries): 10 seconds
  - Finger stabilization interval: 120 seconds

- Kademlia
  - Bucket size: 8
  - Retries before a node is removed: 2
  - Bucket refresh delay: 1000 seconds
  - Network diameter: O(log{2})

- GIA
  - Maximum number of neighbours: 50
  - Minimum number of neighbours: 10
  - Topology adaptation interval: 120 seconds
  - Delay between two update messages: 60 seconds
  - Maximum TTL for sent messages: 10 hops
  - Time before a message is regarded to be lost: 180 seconds
  - Time before a neighbour is regarded to be lost: 250 seconds
  - Interval of issuing tokens: 5 seconds
  - Time take to send new key list to neighbours: 100 seconds

- Underlay backbone parameters
  - Number of backbone routers: 1
  - Number of overlay access routers: 2

The default parameters assumed above were suggested and assumed in [1], [36], as a realistic and practical setting for simulating P2P networks when using typical computing resources. We have made the same assumptions, to make our results comparable with those of [1] and [36].

4.1.4 Statistical analysis

Any simulation that uses random input data generates random output data, so to get final results one needs to analyze output data statistically. Only final results with acceptably small statistical errors can be considered credible. In this thesis, we have tried to reduce statistical errors by repeating each simulation a number of times, for collecting satisfactorily large samples of data. Specifically we collected data from 20 replications and processed them through SPSS, a well known statistical analysis software [16]. Having provided raw output data from simulations, SPSS was used to obtain estimates of the mean, variance, standard error of the mean and then derive the relative error for the

results. The results presented in this thesis have a relative statistical error no larger than 5% at 0.99 confidence level.

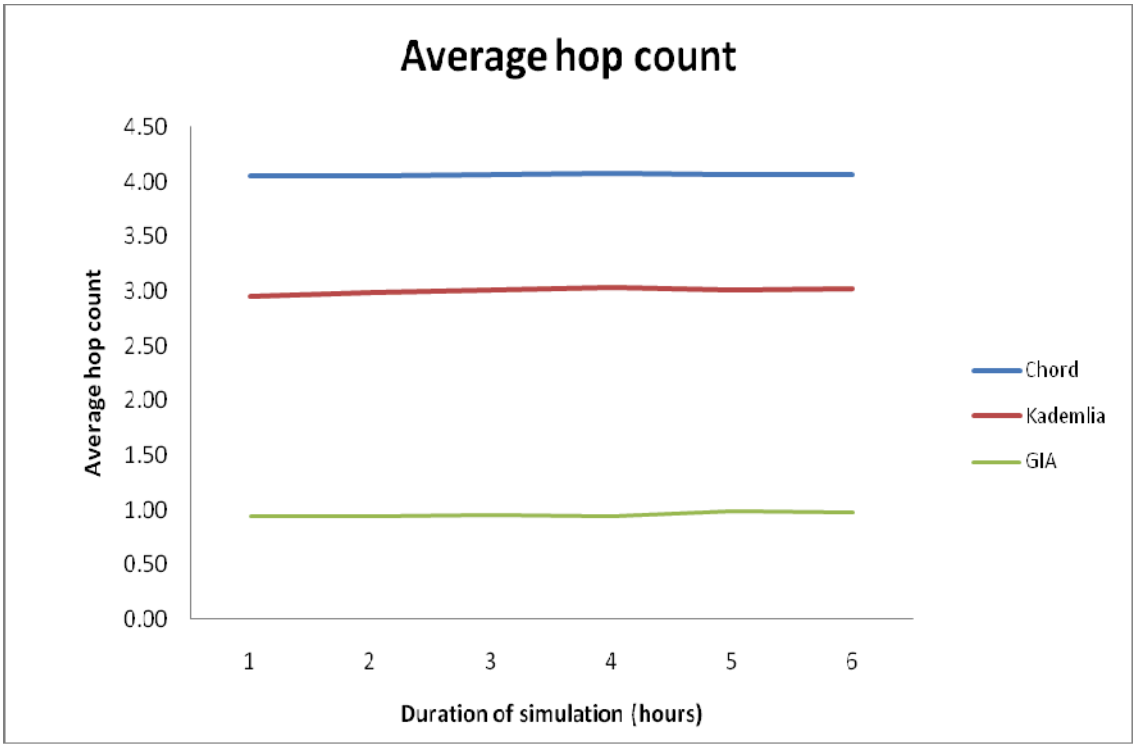


Figure 4.2 A plot of average hop count as a function of simulation durations, assuming constant average activity period equal to 3600 seconds.

Figure 4.2 shows the plot of average hop count obtained for GIA, Kademlia and Chord, if they were simulated over one to six hours. One can see that average hop count remains constant regardless the length of simulation. While this suggests that one could run simulations even of one hour duration to make sure that our simulation results are obtained from sufficiently large samples, we have run all simulations for six hours.

Note that, OverSim goes through a warming up period to populate the simulated network to the designated size. We investigated the time needed for populating networks for their full capacity by identifying the length of time OverSim requires to populate a network of 100 peers. In Figure 4.3, the results show that Kademlia and GIA required between only 9~10 seconds to populate the network to its designated size, while Chord required between 99 to 100 seconds. For the simulation lasting six hours, the warm up period of

100 seconds is not significant, as it is less than 0.5% of the simulation run length. We have concluded that collecting data during 6 hours of simulation will not give biased results in a noticeable way.

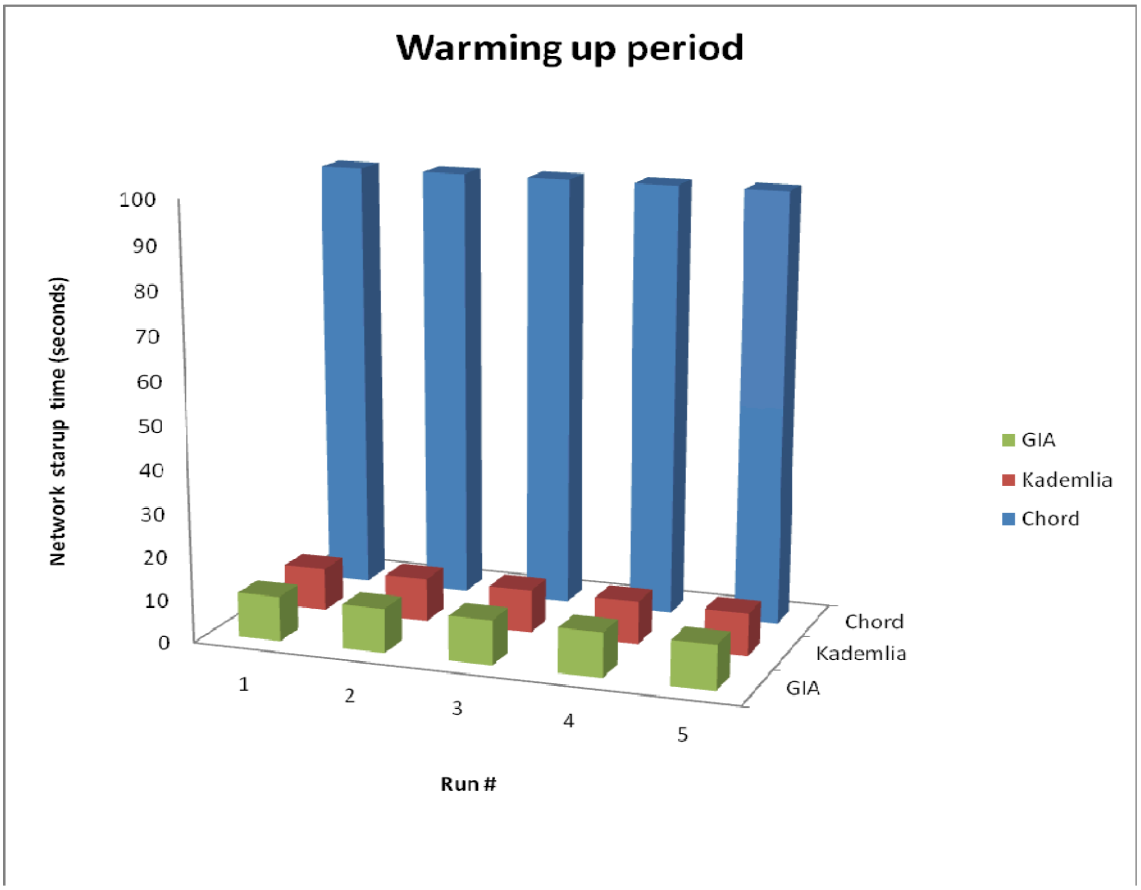


Figure 4.3 A plot of time required to populate network

The file lookup procedures we considered in our simulations are in Figure 4.4. Details of the file lookups in these P2P networks can be found in Chapter 2.

4.2 Simulation Studies and Results

We report here four studies in which the effects of churn in Chord, Kademlia, and GIA were considered and mutually compared. All results are presented with their confidence intervals; if curves are depicted without confidence intervals it means that they were negligibly small to be shown.

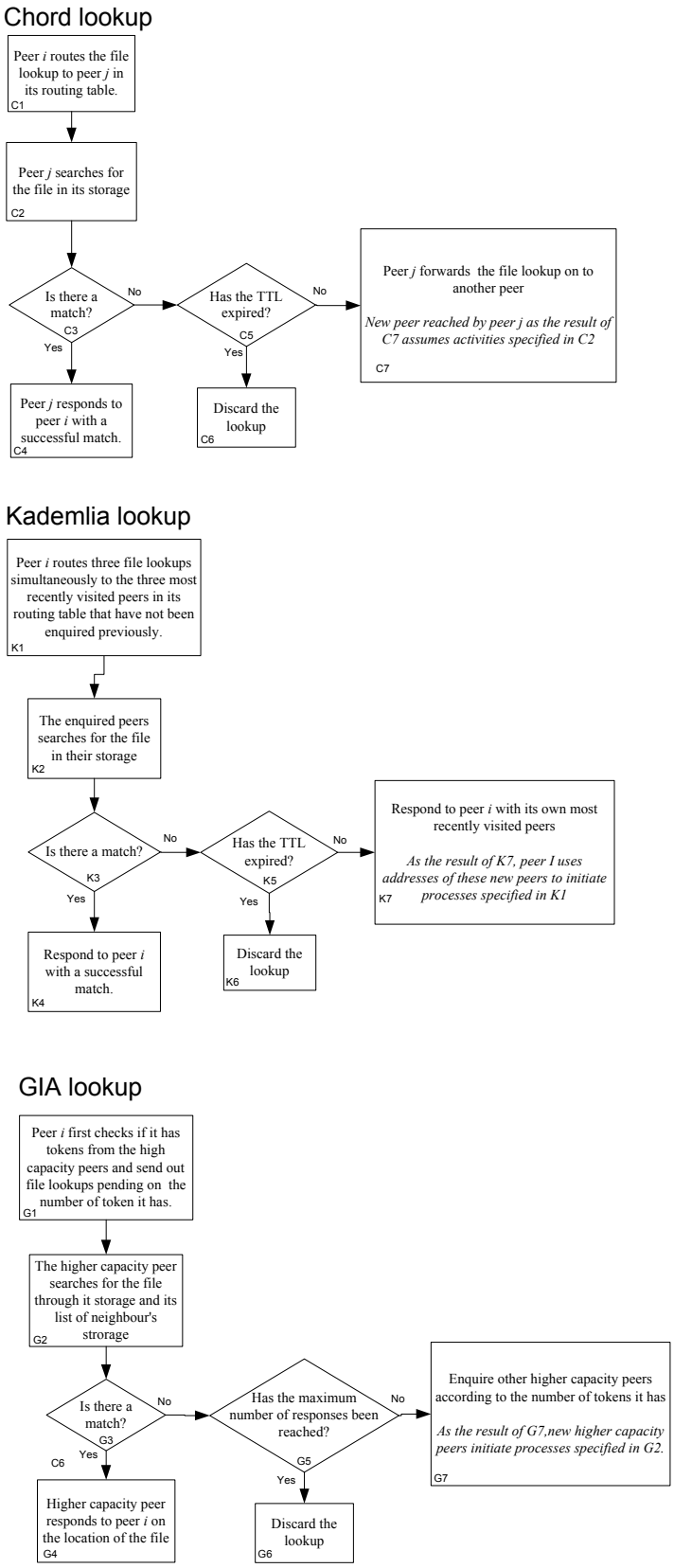


Figure 4.4 The file lookup procedure in Chord, Kademlia and GIA.

4.2.1 Performance of P2P networks under iterative and recursive routing

In this section we have examined the tradeoffs between the file lookup efficiency and success rate under iterative and recursive routing in Chord and Kademlia.

Simulated Scenario: The network goes through stabilization processes until it reaches its maximum capacity of 100 peers. One important feature of file lookup mechanisms is the key distribution factor defined as the percentage of all available keys (file identifiers) which is assigned to a joining peer. In literature, authors were assuming such values for key distribution as 5% and 10% [41], [1], and file lookup intervals both below one minute and larger than three and even ten minutes [32], [36], [52]. Assuming too small values for key distribution factor makes search for file less efficient. On the other hand assuming too short file lookup intervals could result in extremely long simulation runs, as very large number of events would be generated. If the file look up intervals are assumed very long then it could be difficult to collect a sufficiently large sample of output data in a fixed time interval.

Taking this into account, we assumed:

- While a peer joins the network it is assigned to hold 10% of available keys which means that it is allocated with ten random keys from a group of 100 keys.

- When the network is established peers initiate file lookups every 60 seconds. (except in Section 4.2.4).

Each peer stays connected through its assigned activity duration and the network waits until sleeping duration of any previously connected peer expires before adding a new peer to the network.

Hypothesis: Iterative routing will consume more bandwidth with increased latencies in file lookups. Recursive routing will have a lower delivery ratio, but will conserve more bandwidth.

The feature modified in this scenario is the routing method: Iterative versus Recursive. File lookup in a structured P2P network can be routed with the iterative or recursive method. The results shown in Figure 4.5 indicate that both routing methods are not greatly affected by the intensity of churn because the trends remained rather constant throughout different average activity periods. Overall, recursive routing was able to resolve file lookups with fewer hop counts. Recursive routing in Kademlia shows the best performance in the average latency at 900 seconds of average activity period, which is around 0.05 milliseconds.

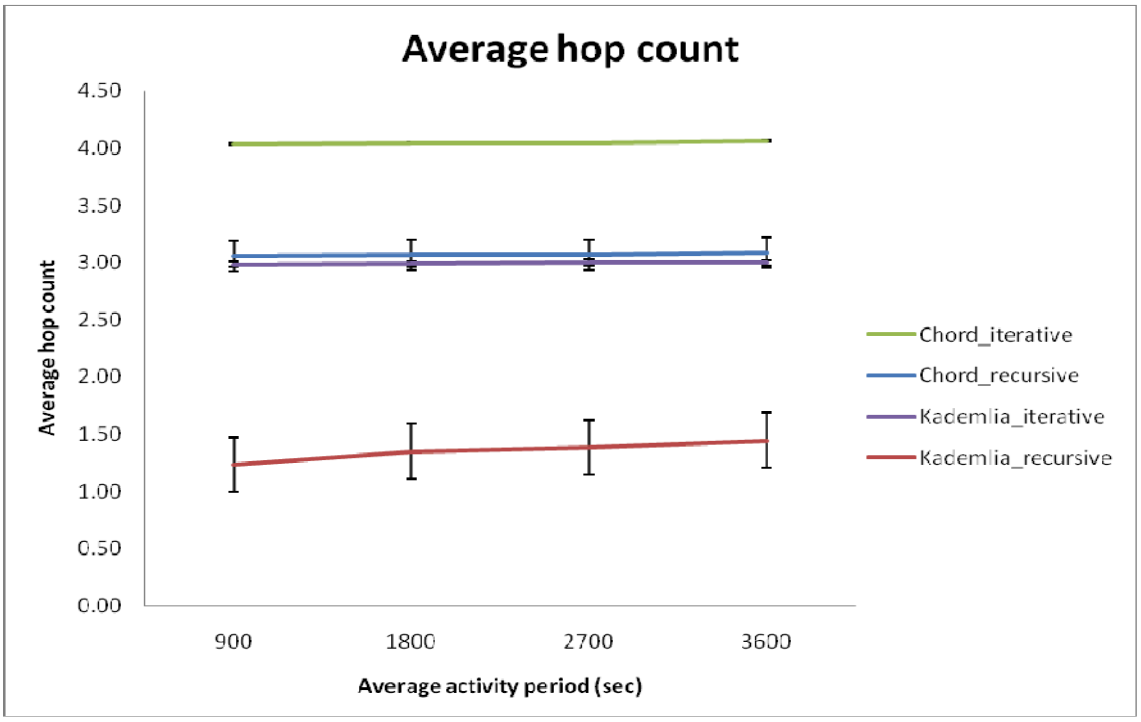


Figure 4.5 A plot of the average hop count for the iterative routing and the recursive routing

In recursive routing, file lookups are only sent to the immediate peer in the routing table which also mean that each peer needs to process fewer number of requests compared to the iterative routing method. As seen in Figure 4.6, recursive routing has achieved smaller latencies overall, and the results were constant throughout different average activity periods. The results of recursive routing showed about 240% improvement in Chord and about 400% improvement in Kademlia. Kademlia uses the XOR-based metric topology which is symmetric and allows peers to receive lookups from approximately the same

number of neighbours stored in their routing tables. Kademlia can send a lookup to any node within an interval and allows the source node to select the best route with the lowest latency.

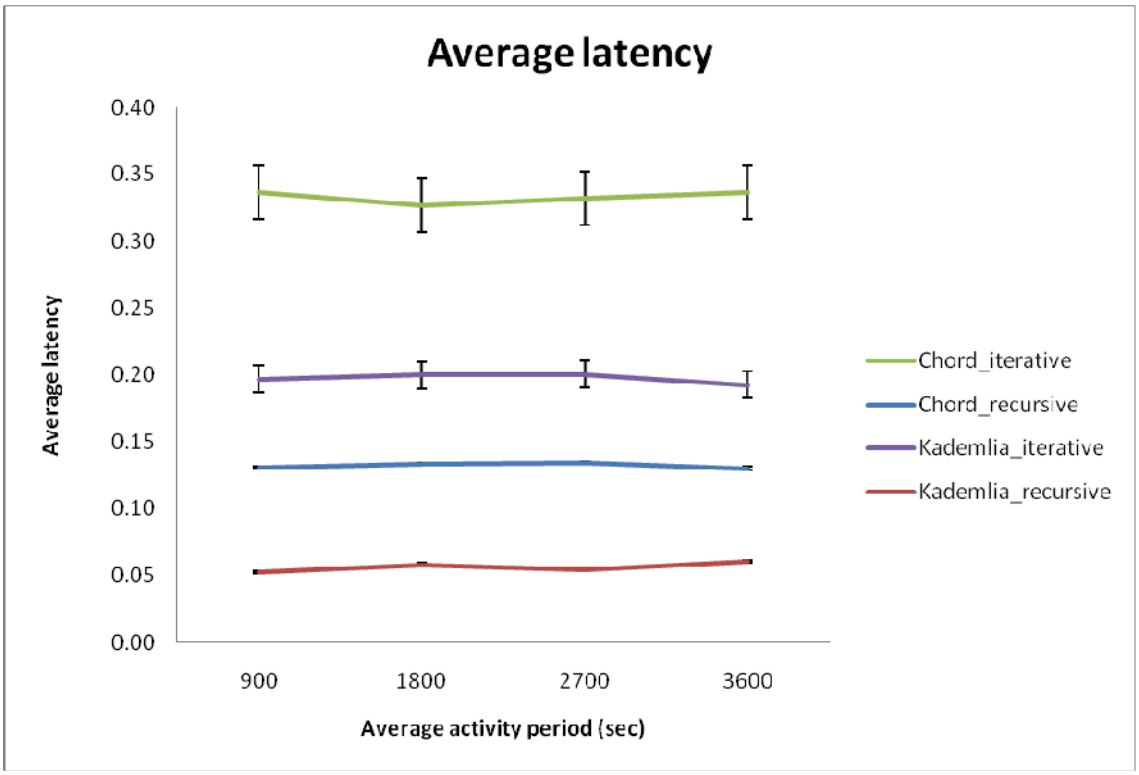


Figure 4.6 A plot of the average latencies for the iterative routing and the recursive routing

As seen in Figure 4.7, the average bandwidth consumption in Chord remains constant, unaffected by the average activity periods. There was a slight increase during lower churn rates where there are more file lookups. Recursive routing Chord performed more efficiently at consuming around 37,000 messages per simulation run of six hours, while iterative routing consumed around 50,000 messages per simulation run of six hours. Iterative routing in Kademlia performed more efficiently in lower churn rates, and showed 77% increase in bandwidth consumption in high churn rate scenarios. Recursive routing in Kademlia offers the lowest bandwidth consumption. Kademlia appends network management messages with file lookups and we can see here the amount of bandwidth conserved was significant compared to Chord.

To conserve bandwidth consumption and reduce latencies, the recursive routing sacrifices the delivery ratio. Delivery ratio represents the success rate of messages arriving to the destination. A low delivery ratio means that file lookups might not have got resolved.

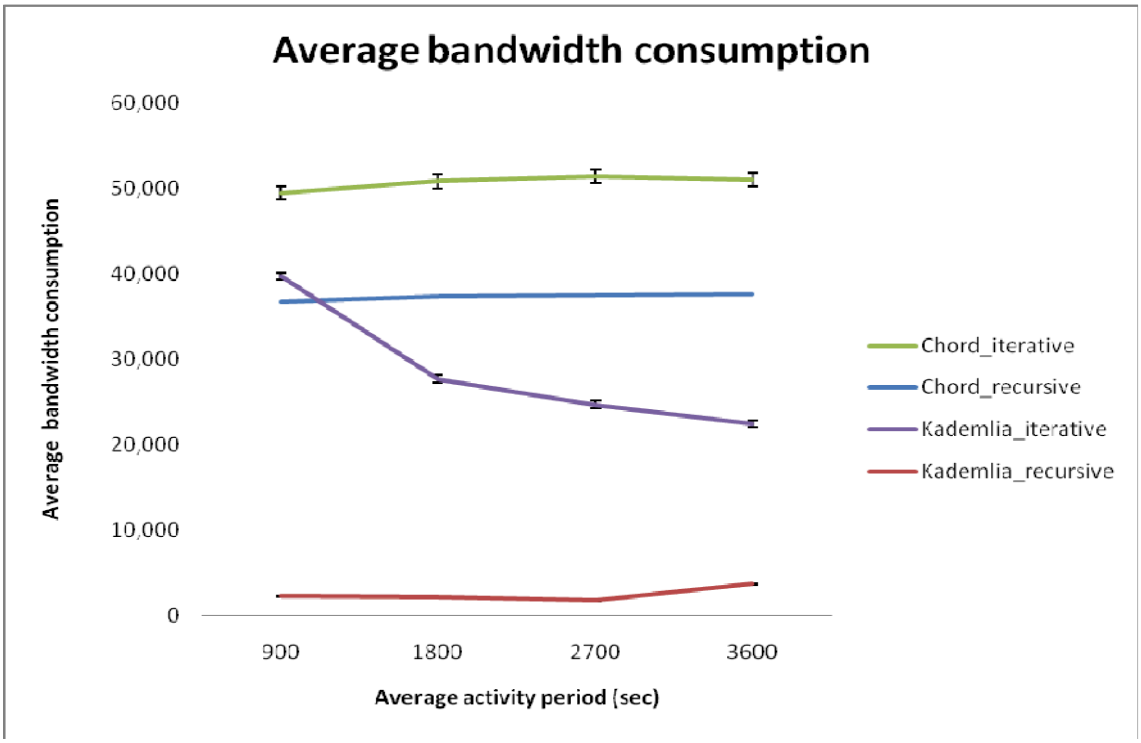


Figure 4.7 A plot of average bandwidth consumption for the iterative routing and the recursive routing

From Figure 4.8, the results show that the iterative routing in Kademlia is the most efficient in maintaining a 99% delivery ratio throughout different churn rates. On the other hand, the recursive routing shows a steady decreasing trend with higher churn rates where the delivery ratio dropped from 28% to 14 %. The recursive routing in Chord is about 10% more efficient than the iterative routing throughout different churn rates. The results for Chord show that both routing methods pose a decreasing trend in efficiency as churn rates increase. Kademlia networks use the tree structure and if one of the higher level nodes disconnects, its leaf nodes will lose connection to other branches. In the recursive routing if the source node does not know where the file lookup request has been lost, it will not know when to start to look for another higher level node. It will forward another request to the following peer in its routing table, instead.

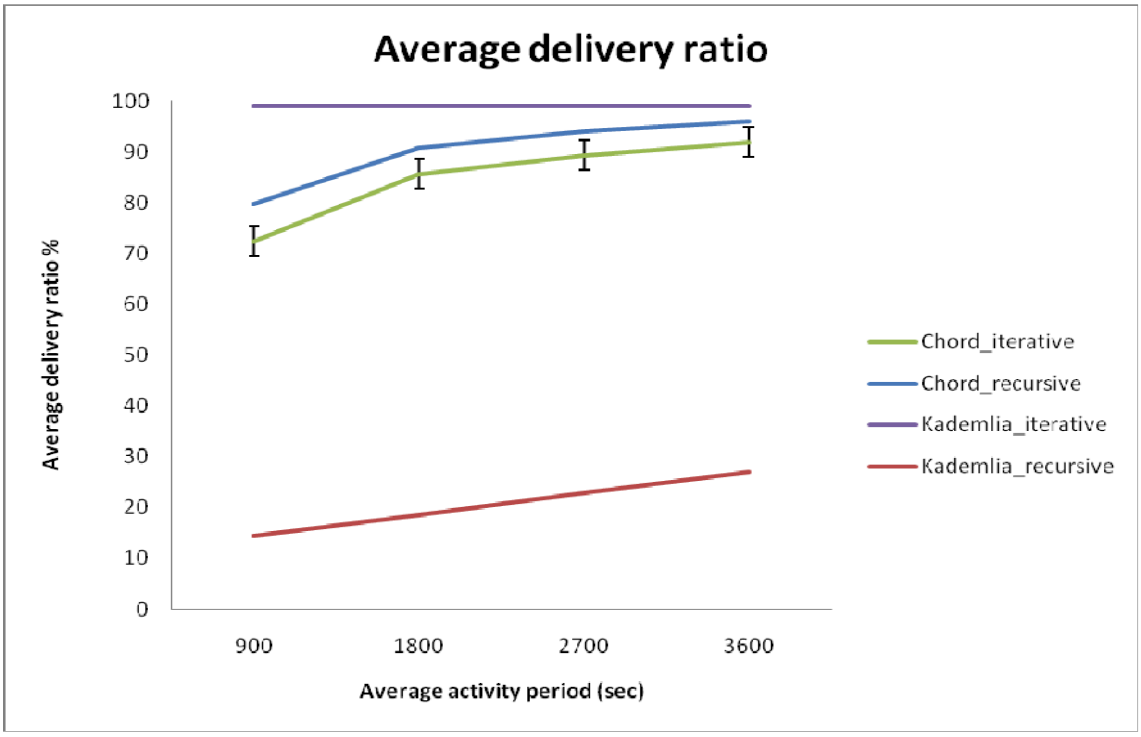


Figure 4.8 A plot of average delivery ratio for the iterative routing and the recursive routing

4.2.2 Performance evaluation of Chord, Kademlia and GIA under higher Churn rates

In this study we have examined the baseline performance of the P2P networks under the effects of churn, which are represented by the different average activity periods. To simulate these effects we have set up an average activity period at 900, 1800, 2700 and 3600 seconds.

Simulated Scenario: The network goes through stabilization process until it reaches its maximum capacity of 100 peers. As previously, we are assuming that as a peer joins the network, it is allocated ten random keys from a group of 100 keys. Each peer stays connected through its assigned activity duration and the network waits until previously connected peer's sleeping duration expires before adding a new peer to the network.

Hypothesis: GIA will perform more efficiently in resolving file lookups compared to structured protocols, and require less bandwidth to sustain churn.

We can conclude from the results shown in Figure 4.9 that the both structured and unstructured networks are able to maintain the quality of resolving file lookups under the effects of churn. Chord had an average of 4 hops, Kademlia had an average of 3 hops and GIA at around 1.3 hops. GIA had better results because GIA networks use 'one hop' structure, where each node maintain a list of keys (files) from each of its neighbours. When a node receives a query, it can respond not only with matches from its contents but also provide matches from the content offered by all of its neighbours.

The results in Figure 4.10 show that GIA consumes the least bandwidth with a decreasing trend as average activity period increases. GIA peers rely on the high capacity peers to resolve their file lookups because high capacity peers can store a lot more lists of keys from a larger set of neighbours. Hence, GIA file lookups tend to be resolved with fewer attempts. Kademlia uses more bandwidth at higher churn rate (average activity period of 900 seconds) at around 40,000 messages per simulation run of six hours, but is able to reduce the bandwidth consumed in lower churn rates.

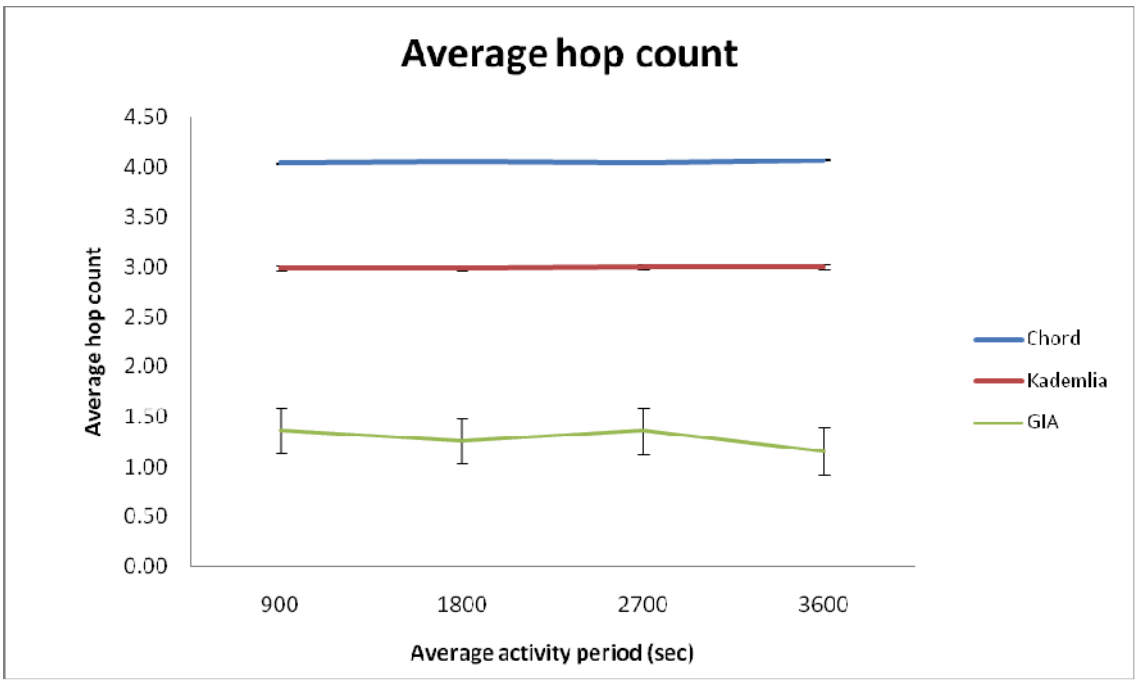


Figure 4.9 A plot of average hop count under the effects of churn

This decreasing trend at the average activity period of 3,600 seconds was around 22,000 messages per simulation run of six hours. Chord was able to maintain constant bandwidth

consumption at around 50,000 messages per simulation run of six hours through high churn rates. At higher churn rates, peers require more network management updates to maintain their routing tables.

From Figure 4.11 we can see that both structured networks have very small latencies compared to GIA. The results from the structured network varied between 0.19 and 0.33 milliseconds. Results from GIA networks show a linear increase as churn rates decrease, with results rising from around 4 milliseconds at 900 seconds of average activity period to 59 milliseconds at 3,600 seconds of average activity period. In GIA networks, all file lookups are sent to the higher capacity nodes, and longer delays in replies can be expected. The linear increase in latency with increasing average activity period is caused by the increase of the number look-up messages flowing around the network.

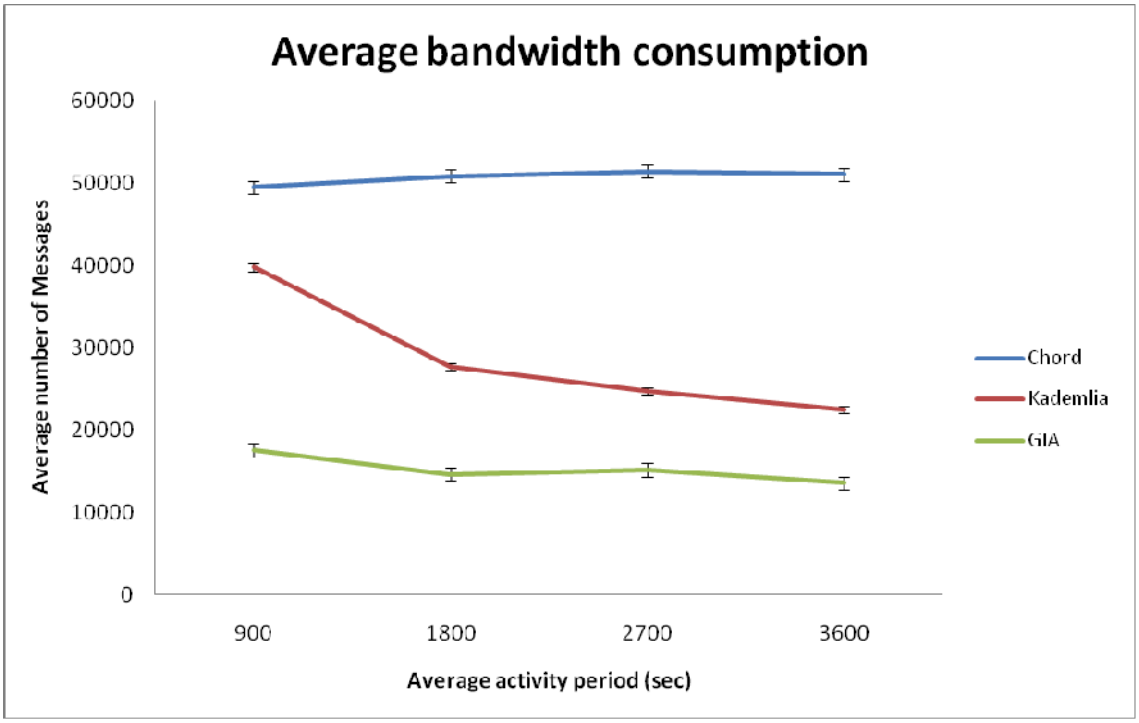


Figure 4.10 A plot of the average bandwidth consumption under the effects of churn

4.2.3 Investigation of scalability of Chord, Kademlia and GIA

In this section, we have examined the robustness of the performance of P2P networks, and their scalability in particular. In this study, the network size is set to grow from the default size of 100 nodes to 1,000 nodes. The goal is to create more disturbances to the networks by simulating more frequent occurrences of peers joining and leaving network.

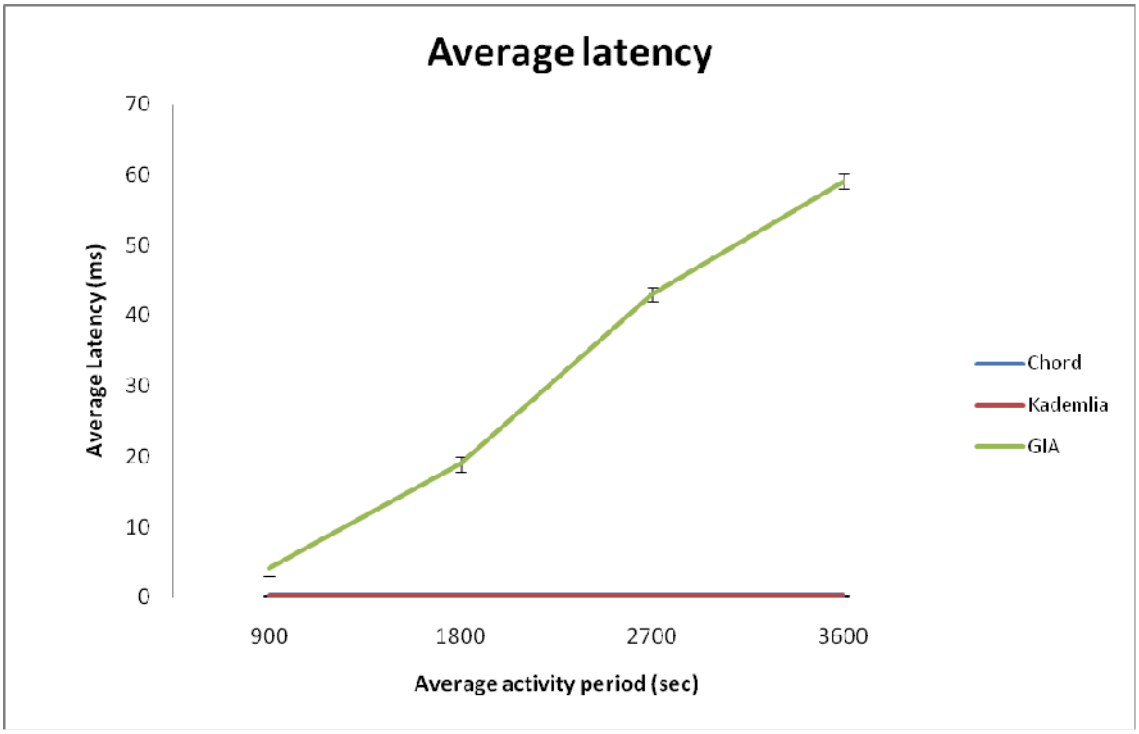


Figure 4.11 A plot of average latency under the the effects of churn

Simulated Scenario: The network goes through stabilization processes until it reaches its maximum capacity of 1,000 peers. As previously mentioned, while each peer joins the network, they are allocated 100 random keys from a group of 1,000 keys. Each peer stays connected through its assigned activity duration and the network waits until previously connected peer's sleeping duration expires before adding a new peer to the network.

Hypothesis: GIA will scale up more efficiently with its size, and would consume less bandwidth compared to structured P2P networks.

As seen in Figure 4.12, the P2P networks show a constant trend throughout different churn rates. There were substantial increases in average hop counts across larger networks compared to results with smaller networks. Chord with 1,000 peers has an average of 5.7 hops. Kademlia of 1,000 peers has an average of 4 hops. GIA with 1,000 peers performs better compared to structured networks with around 1.8 hops. Compared to the results from study 4.2.2, average hop counts in Chord increase by about 43%, while in Kademlia and GIA, the average hop counts increase by about 33%. All three networks seem to adapt reasonably well if network size grows ten times bigger.

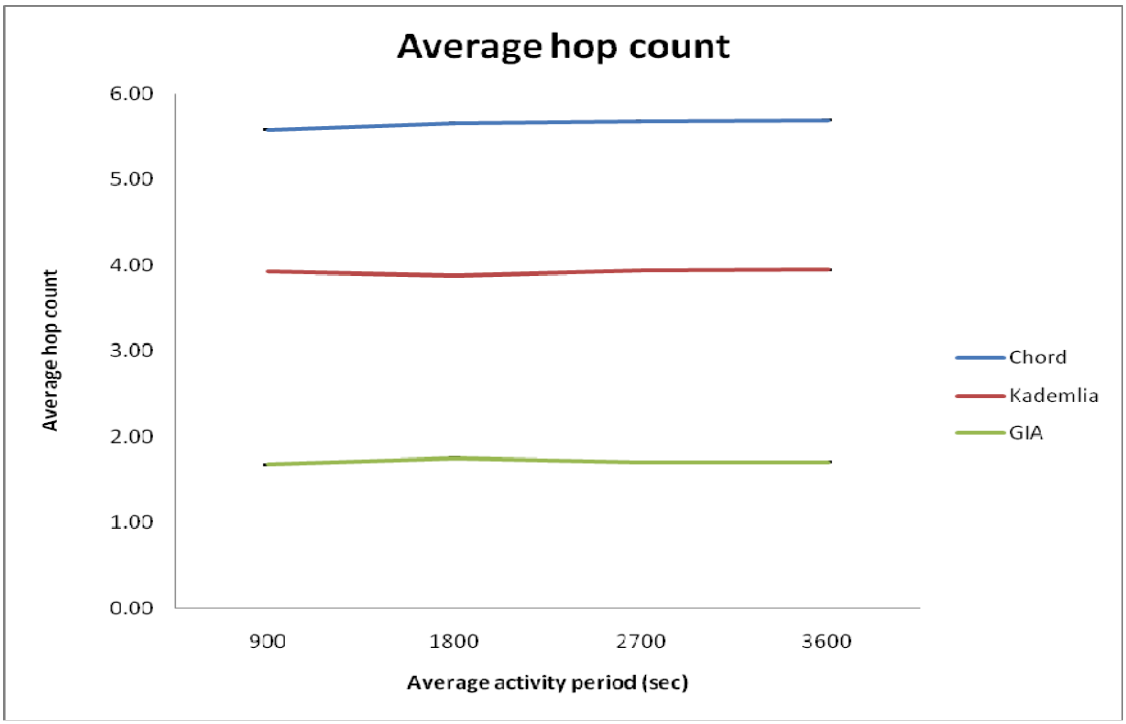


Figure 4.12 A plot of average hop count in a network of 1,000 peers

As seen in Figure 4.13, if the number of peers grows to 1,000, both structured networks maintain very small latencies compared to those of GIA, as they range from 0.3 to 0.43 ms throughout different churn rates. Peer nodes in the structured networks share equal amounts of responsibilities in resolving file lookups, and as we can see, the latencies accumulated are very small. Contrary to this, the larger GIA network actually performed better than the smaller network. If the churn rate lowers below the value corresponding to 1800 seconds of average active period duration, the larger network average latency grows

faster. In the considered cases, the latencies of the GIA network range between 1 and 25 milliseconds. The results from Figure 4.10 show that for the network size of 100, the latencies of GIA were between 4 and 59 milliseconds. This is because in the larger networks more peers are connected to a high capacity node and the chance of having a resolved lookup within the group is higher compared to a smaller network where the groups of nodes are smaller. Hence, searches in larger networks are likely to take less time.

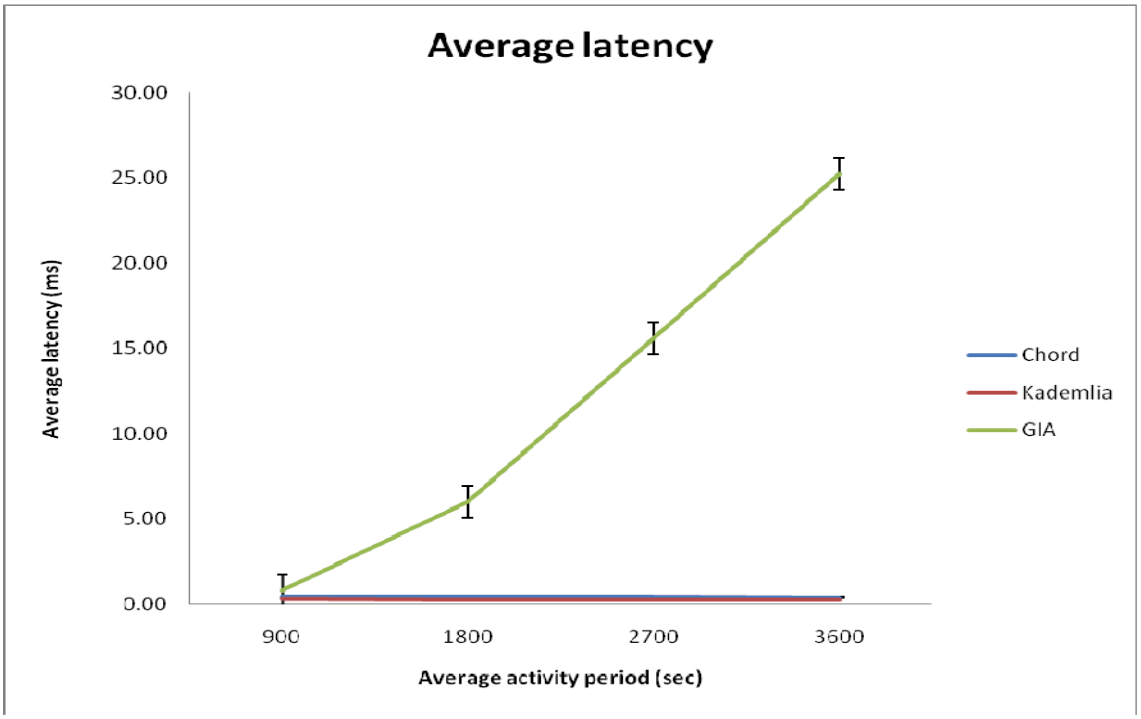


Figure 4.13 A plot of average latencies in a network of 1,000 peers

Chord was able to maintain constant bandwidth consumption at around 70,000 messages per simulation run of six hours for churn rates associated with average activity periods above 1,800s. Compared to the results from Figure 4.10, Chord only used 40% more bandwidth to sustain a network with 1,000 nodes. Chord maintains its routing table periodically and distributes the routing information to nodes in its routing table. This implementation creates overhead in the network and as we can see when the average activity period increases, the amount of file lookups also increases, and adds to total bandwidth consumption.

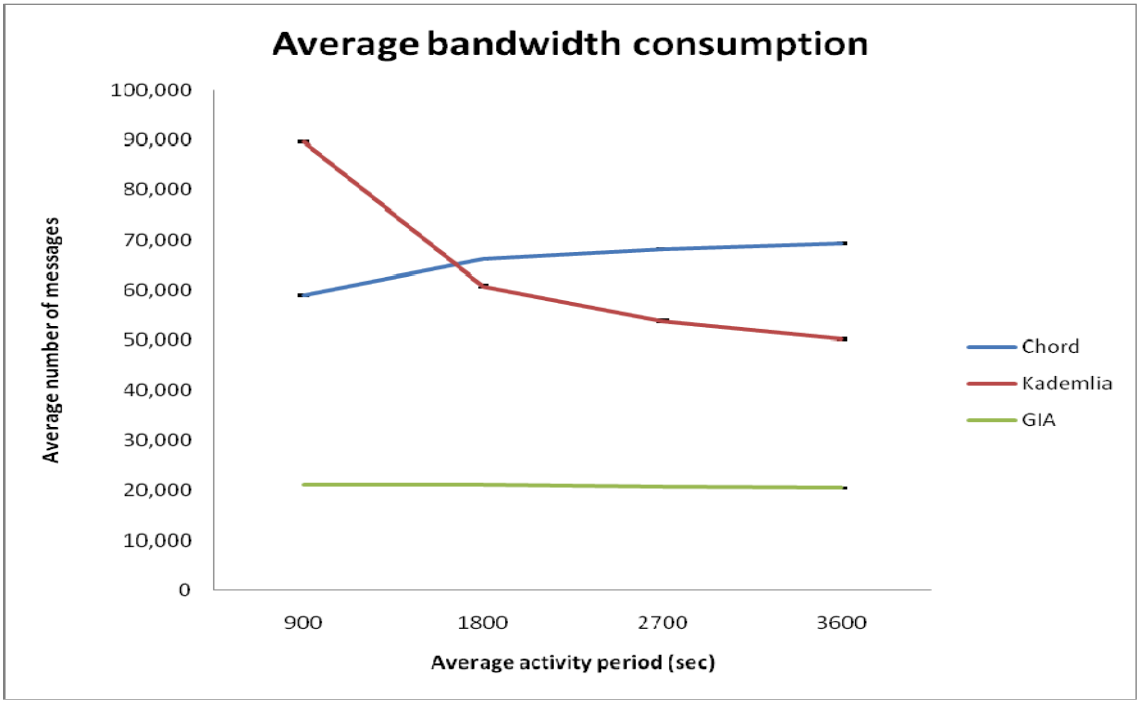


Figure 4.14 A plot of average bandwidth consumption in a network of 1,000 peers

Kademlia presents a similar trend; see Figure 4.10 and Figure 4.14. There is a significant decrease in bandwidth consumption when the average activity period lowers from 900s to 1,800s, with the average number of messages dropped from 90,000 to 60,000. Kademlia requires 125% more bandwidth to sustain churn in the larger network of 1,000 nodes. A peer in Kademlia updates its routing table by appending network management messages on file lookups. As we can see in Figure 4.14, when the network becomes less dynamic and the number of file lookup increases, the bandwidth consumption of Kademlia drops significantly.

GIA shows the least increase in bandwidth consumption. It is able to maintain the network at bandwidth consumption of around 21,000 messages, compared to the average results of around 16,000 messages in the smaller network. As described earlier, GIA peers rely on high capacity peers to resolve file lookups and in larger networks, with more search messages, the bandwidth conserved is significant.

GIA implements a topology adaptation scheme: a GIA peer periodically assesses if the high capacity peer it is connected to is still a high degree node and that low capacity peers

are within short reach of the high capacity peer. Each GIA peer assesses the connection and derives a level of satisfaction. This is a value ranging between 0 and 1; a value of 0 means that the node is dissatisfied and a value of 1 means that the node is satisfied. The level of satisfaction is determined by the set of neighbours that the peer has and if they are sufficient to satisfy its file requests.

In Figure 4.15, the satisfaction level of the 1,000 node and 100 node GIA networks are presented. Both networks show an increase in the satisfaction level with a less dynamic network. Peers in the larger network have better average satisfaction level compared to peers in a smaller network. The satisfaction level of a peer in the larger network ranges between 33% and 53%, while the smaller network ranges between 31% and 44%. This is by the fact that in the larger network, peers will have more neighbours, so the satisfaction level is likely to be higher.

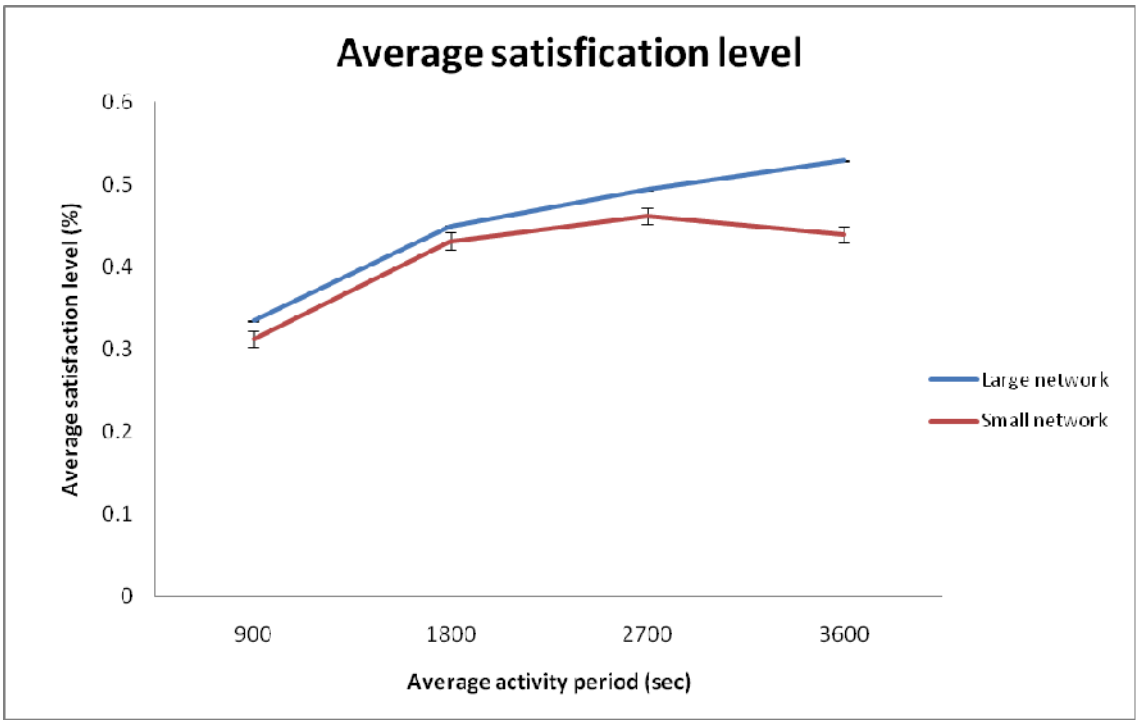


Figure 4.15 A plot of average satisfaction level in GIA in a network of 1,000 peers

4.2.4 Performance of Chord, Kademlia and GIA under higher search frequencies

In this section, we have examined the effects of high search rate on the performance of P2P networks, and investigate their ability to resolve simultaneous lookups. In these experiments, the search frequency was assigned to initiate at every 15/30/45/60 seconds.

Simulated Scenario: The network goes through stabilization processes until it reaches its maximum capacity of 100 peers. When a peer joins the network, it is allocated with ten random keys from a group of 100 keys. When the network is established all peers initiate file lookup processes of duration 15, 30, 45 or 60 seconds. Each peer stays connected through its assigned activity duration and the network waits until previously connected peer's sleeping interval expires before adding a new peer to the network.

Hypothesis: GIA will perform more efficient when dealing with file lookups as it will use less bandwidth, but will suffer higher delays compared to structured protocols.

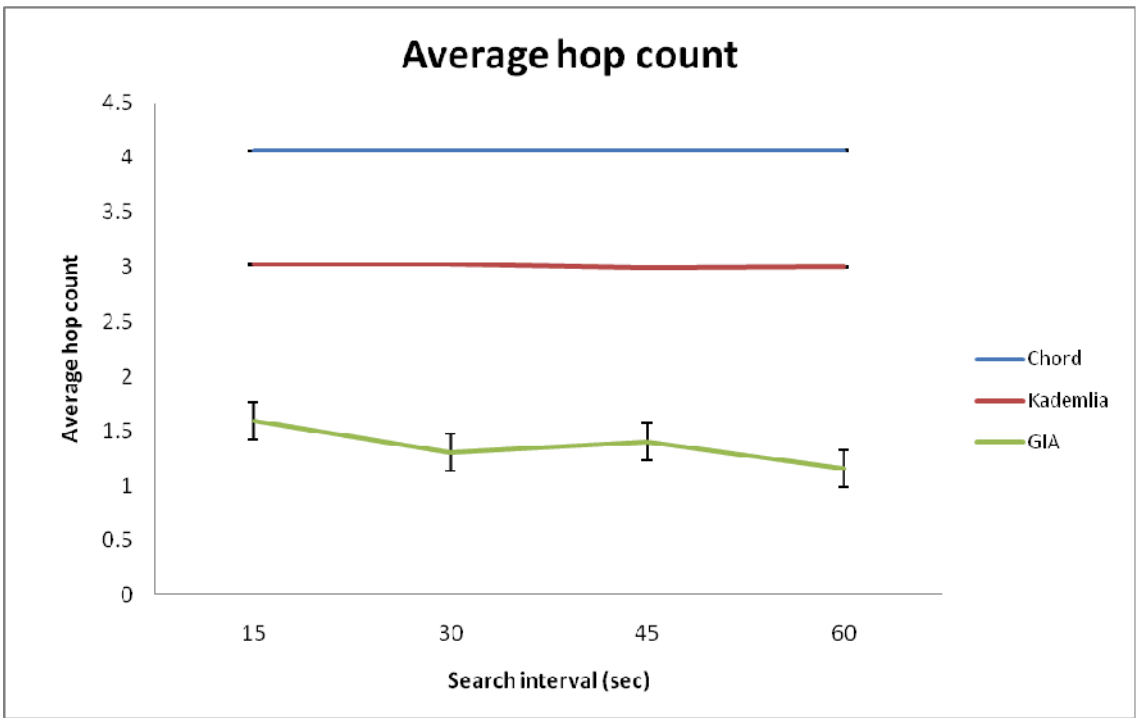


Figure 4.16 Average hop count as a function of search frequency

As seen in Figure 4.16, structured networks are able to maintain a constant average hop count throughout different search frequencies, with Kademlia's count of 3 hops and Chord's count of around 4 hops. GIA experiences a 35% increase in hop counts at shorter search interval of 15 seconds, and equals to1.5, compared to the default search interval of 60 seconds having 1.2 average hop count. Compared to the results from Figure 4.7 which has a search interval of 60 seconds, all three networks adapted well to a busier situation.

As seen in Figure 4.17, both structured networks maintain a small latency ranging between 0.17 to 0.3 ms throughout different search frequencies. In structured networks all peers share equal amount of responsibilities in each other's resolving file lookup and the latencies incurred in this scheme are low if compared with GIA networks. GIA networks accumulate a high latency of 114 milliseconds at the search rate of 15 second intervals. The trend decreases with decreasing search rates, and at the search rate of every 60 seconds per search the latency was around 60 milliseconds. GIA networks rely on high capacity peers to resolve file lookups, and when the high capacity peers have to resolve a lot more lookups in the case of shorter search intervals, the latencies are a lot higher. The latencies decrease as the number of search messages reduces in longer search intervals.

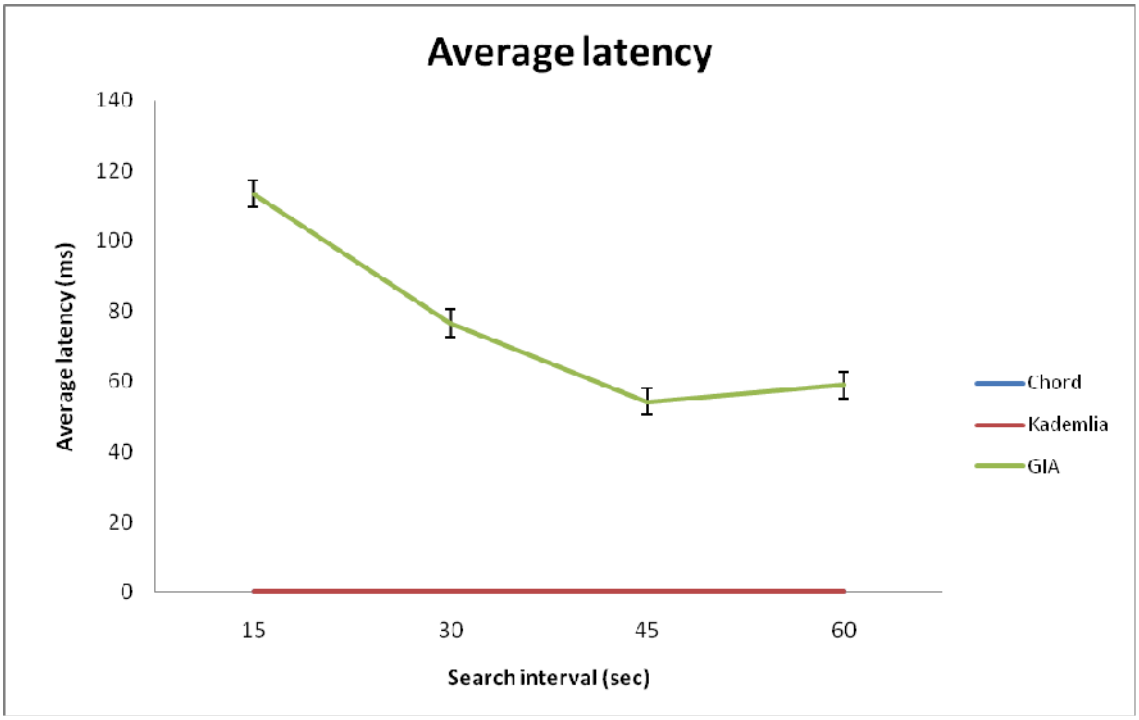


Figure 4.17 A plot of average latencies as a function of search frequency

As seen in Figure 4.18, all three networks show that they consume more bandwidth if search intervals are shorter. Chord has the highest bandwidth consumption overall, ranging from 65,000 to 50,000 messages. Both Chord and Kademlia consume more than 65,000 messages per simulation of six hours for 15 second search intervals. However, there is a significant drop in the rate of growth after the search rate lowers to one per every 30 seconds and less, while Chord has a 17% reduction and Kademlia has a 44% reduction in bandwidth consumption. GIA shows a steady decrease from producing 27,000 messages for the search interval of 15 seconds, to 13,000 messages at a search interval of 60 seconds.

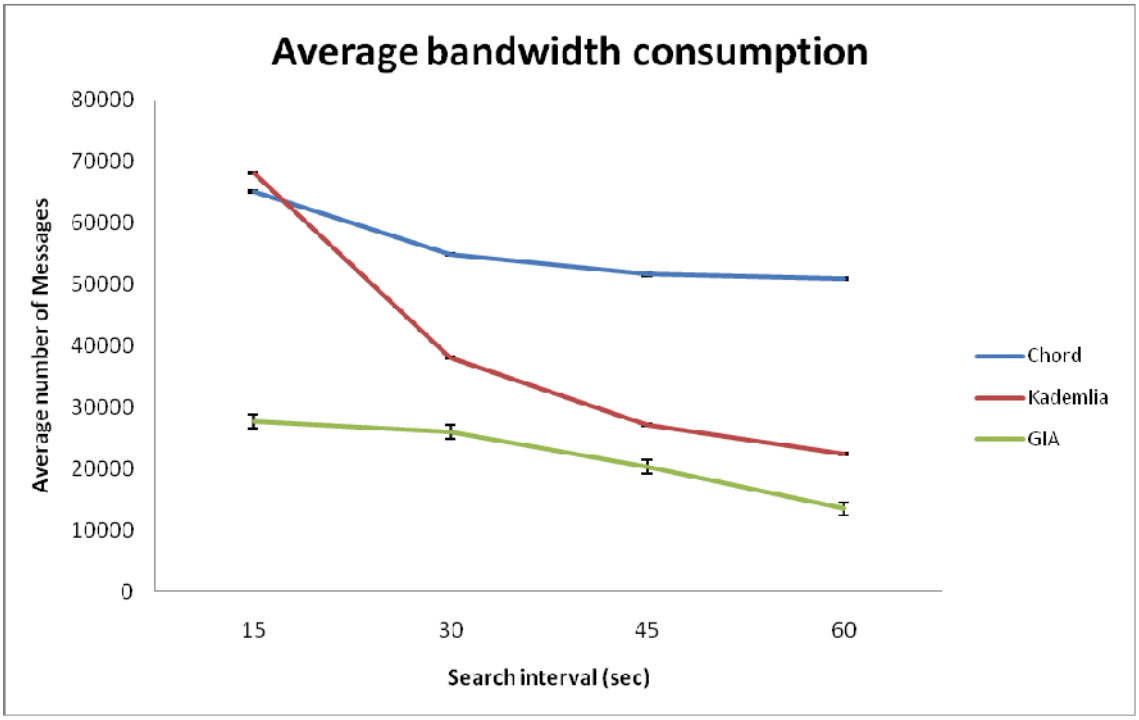


Figure 4.18 A plot of average bandwidth consumption as a function of search frequencies

4.3 Summary

In this chapter we have presented the resutls of our analysis on the performance of Chord, Kademlia and GIA during their resource discovery phase under churn. Prior to the simulation studies we justified the selection of 6 hours as the duration of our simulation runs. We have also shown that the effects of networks' warm up period on our results are

negligible. Four simulation studies were designed to evalute the performance of the three P2P networks. In our first study we have showed that recursive routing consumes smaller bandwidth and involves smaller delays. However, the delivery ratio of messages in recursive routing is unsatisfactory. In the second study, we have compared the performance of Chord, Kademlia and GIA. We have studied the effects of churn by setting different lengths of average activity periods. The results show that GIA consumes smaller bandwidth and is able to resolve file lookups with better results. However, GIA suffers from high latencies compared to Chord and Kademlia. In the thrid study, we have investigated the scabilities of the P2P networks. The results show that all the P2P networks can operate efficiently as their sizes grow (at least in the considered case, when the size grows form 100 to 1,000 peers). GIA performed even better with smaller latencies and produces higher satisfaction level if the number of peers increases. In the fourth study, we have investigated the effects of high search fequencies on the P2P networks. The results show that all three types of P2P networks decrease the quality of their performance with higher search rates. GIA was able to consume smaller bandwidth and resolve lookups with better results than Chord and Kadmliea, but its latencies were much higher.

Chapter 5

Conclusions

The volume of P2P traffic has increased significantly in the past few years and it is accounted to be the main type of traffic for ISPs all around the globe. P2P networks allow direct connections between clients without the need for central servers and the size of these networks is quickly increasing. Without efficient network management schemes and resource discovery algorithms the performance of these P2P networks could deteriorate, deteriorating the quality of other services as well.

In Chapter 2, we have presented the architecture of Chord, Kademlia and GIA. We have outlined the network management schemes and resource discovery phases of each of these networks. Unlike a traditional server and client network, P2P peers can join and leave a network unexpectedly, and the network becomes unstable. Without a central management entity, individual peers are responsible for updating their own routing table and resolving file lookups between each other. During a file lookup, peers will depend on each other when searching for a match of the requested data, and because the resources are limited in most peers, an efficient routing scheme is crucial. In a large dynamic network if efficient mechanisms are not implemented, these issues can escalate to halt operations in a short period of time. Chord and Kademlia are categorized as structured networks that have adopted DHTs. In structured networks there is a set of guidelines which each peer will need to follow to connect to the network, manage its own routing table and handle file lookups. GIA is categorized as a hybrid network under unstructured networks. The characteristics of GIA are that all peers will connect to high capacity nodes and route all file lookups to the high capacity nodes. High capacity nodes will have lists of resource indices from nodes that are connected to them.

In Chapter 3, we discussed our experiment methodology, surveyed P2P simulators, and considered models of churn, as a vital characteristic of P2P processes.

In Chapter 4, we evaluated Chord, Kademlia and GIA through simulation studies. Our results indicate that GIA is able to perform more efficiently because it is able to resolve file lookups with smaller hop counts and consumes less bandwidth. GIA peers connect themselves to other GIA peers with higher capacity and rely on them to resolve file lookups. This implementation can locate a match that is closer to the enquiring peer because high capacity peers can not only compare the file lookup with their own contents, but also contents from all of their neighbours. The other benefit of relying on high capacity peers is the conserved bandwidth, because lookups are likely to be resolved by high capacity peer and enquiring peers will not need to enquire other peers. The weakness of this version of P2P networks is that it involves higher latencies than Chord and Kademlia because high capacity peers of GIA need to service all their neighbours. However, from our results we can conclude that the latencies are not substantial and peers are not likely to suffer from this reason.

5.1 Future Work

There is scope for further work in implementing more realistic churn models. One of the characteristics of real P2P peers is the 'free riding' scenario. 'Free riding' is a phenomenon describing the selfishness of peers where they are not prepared to share their resources, at the same time taking advantage of the resources shared by other peers. The Pareto distribution we have used as a model for "activity" and "sleeping" phases could be extended to provide models for such peer behaviour as well.

The unstructured network GIA implemented in OverSim could be further extended. The flow control mechanism in a GIA peer is supposed to assess the peer's available resources and adjust the rate of token issuing so the peer do not get overwhelmed with incoming file lookups. In OverSim the flow control mechanisms periodically issue tokens to neighbours without assessing the source peer's available resources.

We showed that the unstructured P2P networks are better than structured P2P networks, and further research could be aimed at finding the best unstructured P2P networks. For example, by considering the performance of newer versions of Gnutella.

This research project has not taken into account security issues related to P2P systems. It is generally accepted that structured P2P networks are more immune to security threats than unstructured P2P networks. Thus, selection of a specific type of P2P network in practical situations can require additional compromise between quality of performance and level of security.

References

[1] *OverSim: The Overlay Simulation Framework*. Available from: http://oversim.org/ [visited 2010 06/04]

[2] *OMNeT++*. Available from: http://www.omnetpp.org/ [visited 2010 06/04]

[3] *The Network Simulator - ns-2*. Available from: http://www.isi.edu/nsnam/ns/ [visited 2010 06/04]

[4] *PeerSim: A Peer-to-Peer Simulator*. Available from: http://peersim.sourceforge.net/#docs [visited 2010 06/04];

[5] *NeuroGrid | P2P Search*. Available from: http://www.neurogrid.net/php/index.php [visited 2010 06/04];

[6] *p2psim: a simulator for peer-to-peer protocols*. Available from: http://pdos.csail.mit.edu/p2psim/ [visited 2010 06/04];

[7] *The ns-3 network simulator*. Available from: http://www.nsnam.org/ [visited 2010 06/04];

[8] *Napster*. Available from: http://www.napster.com [visited 2010 06/04];

[9] *Gnutella - A Protocol for a Revolution*. Available from: http://rfc-gnutella.sourceforge.net/ [visited 2010 06/04];

[10] *INET Framework*. Available from: http://inet.omnetpp.org/ [visited 2010 06/04]

[11] *The Network Simulator ns-2: Topology Generation*. Available from: http://www.isi.edu/nsnam/ns/ns-topogen.html [visited 2010 06/04]

[12] *CAIDA : tools : measurement : skitter*. Available from: http://www.caida.org/tools/measurement/skitter/ [visited 2010 06/04];

[13] *Welcome to PlanetSim project.* Available from: http://projects-deim.urv.cat/trac/planetsim/ [visited 2010 06/04];

[14] *KaZaA*. Available from: http://www.kazaa.com/ [visited 2010 06/04];

[15] *RFC 3174 - US Secure Hash Algorithm 1 (SHA1)*. Available from: http://www.faqs.org/rfcs/rfc3174.html [visited 2010 06/04];

[16] *IBM SPSS*. Available from: http://www.spss.com/ [visited 2010 06/04];

[17] K. Aberer and H. Manfred, *Peer-to-Peer Systems*, in "*The Practical Handbook of Internet Computing*". Edited by M.P. Singh, Chapman & Hall/CRC Computer, 2005.

[18] V. Aggarwal, O. Akonjang, A. Feldmann, R. Tashev and S. Mohr, *Reflecting P2P User Behaviour Models in a Simulation Environment*, in *Proceedings of Parallel, Distributed and Network-Based Processing* 2008, IEEE. Pages 516-523

[19] S. Androutsellis-Theotokis and D. Spinellis, *A Survey of Peer-to-Peer Content Distribution Technologies*. Athens University of Economics and Business, 2004. Pages 335-371.

[20] F. Atalla, D. Miranda, J. Almeida, M. Gonçalves, V. Almeida, *Analyzing the Impact of Churn and Malicious Behavior on the Quality of PeertoPeer Web Search*, in *Proceedings of the 2008 ACM symposium on Applied Computin.* Pages 1137-1144.

[21] M. Baker and R. Lakhoo, *Peer-to-Peer Simulators*. University of Reading. 2007.

[22] R. Bhagwan, S. Savage, and G Voelker, *Understanding availability*, in *Proceedings of International Workshop on Peer-To-Peer Systems 2003.* Pages 256-267.

[23] A. Binzenhöfer and L. Kenji, *Estimating Churn in Structured P2P Networks*, in *Proceedings of ITC 2007,* IEEE. Pages 630-641.

[24] A. Binzenhöfer, D. Staehle and R. Henjes, *On the Stability of Chord-based P2P Systems*, in *Proceedings of IEEE GLOBECOM 2005*. Pages 884-888.

[25] R. Brunner, in Master's Thesis "*A performance evaluation of the Kad-protocol*." University of Mannheim. 2006.

[26] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham and S. Shenker, *Making Gnutella-like P2P Systems Scalable*, in *Proceedings of ACM SIGCOMM* 2003. Pages 407-418.

[27] J. Eberspächer, S. Rchollmeier, S. Zöls, G. Kunzmann and L. Für, *Structured P2P Networks in Mobile and Fixed Environments*, in *Proceedings of International Working Conference on Performance Modeling and Evaluation of Heterogeneous Networks 2004*.

[28] P. García, C. Pairot, R. Mondéjar, J. Pujol, H. Tejedor, and R. Rallo, *PlanetSim: A New Overlay Network Simulation Framework*, in *Proceedings of Software Engineering and Middleware 2004*, IEEE. Pages 123-136.

[29] P. Godfrey, S. Shenker, and I. Stoica, *Minimizing churn in distributed systems*, in *Proceedings of ACM SIGCOMM 2006*. Pages 147-158.

[30] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan *Measurement, Modeling, and Analysis of P2P File-Sharing Workload*, in *Proceedings of Special Interest Groupon Operating Systems 2003*, ACM. Pages 314-329.

[31] O. Herrera and T. Znati, *Modeling Churn in P2P Networks*, in *Proceedings of Annual Simulation Symposium 2007*, IEEE. Pages 33-40.

[32] O. Herrera-Ruiz, and T. Znati, *Static Resiliency vs Churn-Resistance Capability of DHT-Protocols*, in Proceedings of *International Society for Computers and their Applications: Parallel and Distributed Computing Systems*, 2005. Pages 141-147.

[33] N. Hung, N. Giang, and T. Yew, *Performance Evaluation of Distributed Hash Table (DHT) Chord Algorithm*, in *Proceedings of International Society of Science and Applied Technologies: Modeling of Complex Systems and Environments 2007*.

[34] E. Klemm, C. Lindemann, and O. Waldhorst, *Relating Query Popularity and File Replication in the Gnutella P2P Network*, in Proceedings of *Measurement, Modeling, and Evaluation of Computer and Communication Systems 2004* (MMB 2004).

[35] N. Kotilainen, M. Vapa, T. Keltanen, A. Auvinen and J. Vuori, *P2PRealm – Peer-to-Peer Network Simulator*, in *Proceedings of Computer-Aided Modeling, Analysis and Design of Communication Links and Networks 2006*. Pages 93-99.

[36] J. Li, J. Stribling, T. Gil, R. Morris, and M. Kaashoek, *Comparing the Performance of Distributed Hash Tables Under Churn*, in *Proceedings of International Workshop on Peer-To-Peer Systems 2004*. Pages 87-99.

[37] P. Maymounkov and D. Mazières, *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*, in Proceedings of International Workshop on *Peer-To-Peer Systems 2002*. Pages 53-65.

[38] A. Medina, A. Lakhina, I. Matta, and J. Byers, *BRITE: an approach to universal topology generation*, in Proceedings of *Modeling, Analysis and Simulation of Computer and Telecommunication Systems 2001*. Pages 346-353.

[39] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman and D. Chalmers, *The State of Peer-to-Peer Simulators and Simulations*, in *Proceedings of ACM SIGCOMM 2007*. Pages 95-98.

[40] D. Nurmi, J. Brevik, and R. Wolski, *Modeling machine availability in enterprise and wide-area distributed computing environments*, in *Proceedings of Euro-Par 2005*. Pages 432-441.

[41] D. Patanroi, and G. Lingappa, *GiaSim: GIA Implementation Using NS2 Simulator*. Iowa State University of Science and Technology. 2004.

[42] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, *Handling Churn in a DHT*, in Proceedings of *USENIX Annual Technical Conference 2004.* Pages 1-14.

[43] S. Saroiu, P. Gummadi, and S. Gribble, *A measurement study of peer-to-peer file sharing systems*, in Proceedings of *Multimedia Computing and Networking 2002*.

[44] S. Sen and J. Wang, *Analyzing Peer-To-Peer Traffic Across Large Network*, in *Proceedings of IEEE/ACM Transactions on Networking 2004*, pages 219-232. 2004.

[45] R. Steinmetz and K. Wehrle, in "*Peer-to-Peer Systems and Applications*." 2005. Springer-Verlag Berlin Heidelberg. Pages 9-16.

[46] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, *Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications*, in *Proceedings of IEEE/ACM Transactions on Networking 2003*, pages 17-32.

[47] D. Stutzbach and R. Rejaie, in the technical report "*Towards a Better Understanding of Churn in Peer-to-Peer Networks*.", University of Oregon. 2004

[48] D. Stutzbach and R. Rejaie, *Understanding Churn in Peer-to-Peer Networks*, in *Proceedings of ACM SIGCOMM: Internet measurement 2006*. Pages 189-202.

[49] D. Tsoumakos, N. Roussopoulous, *Analysis and Comparison of P2P Search Methods*, in *Proceedings of International conference on Scalable information systems 2006*.

[50] D. Yang, Y. Zhang., H. Zhang, T. Wu and H. Chao, *Multi-factors oriented study of P2P Churn*, in International Journal of Communication Systems, 2009. Vol.22, Issue 9, pages 1089-1103.

[51] Z. Yao, D. Leonard, X. Wang, and D. Loguinov, *Modeling Heterogeneous User Churn and Local Resilience of Unstructured P2P Networks*, in Proceedings of *International Conference Network Protocols 2006*. Pages 32-41.

Glossary

Churn: The dynamic variability of the population of peers in P2P networks which is caused by peers joining and leaving the network randomly.

Structured P2P network: A network that implements Distributed Hash Tables and uses consistent hashing to structure its P2P network.

Unstructured P2P network: A network that structures itself without any central entity management.

File lookup: A search request from a peer, enquiring other peers in search of a match on a particular keyword.

Active period: The period of time where the peers stay connected in the network and ready to send or resolve file lookups.

Sleeping period: The period of them where the peers stay disconnected from the network.

Acronyms

P2P – Peer to Peer

DHT – Distributed Hash Table

SHA-1 – Secure Hash Algorithm-1

RPC – Remote Procedure Call

GIA – Gianduia

TTL – Time To Live

GUID – Globally Unique Identifer

Appendix A

Results

Appendix A1

This section gives the numerical results for section 4.2.1.

Results from Iterative routing

One-Sample Statistics

Chord 900	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0345	.01195	.00267
Bandwidth Consumption	20	50648.4113	2319.32573	518.61700
Latency	20	.3364	.02325	.00520

One-Sample Statistics

Chord 1800	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0492	.00557	.00125
Bandwidth Consumption	20	51564.4327	1148.10467	256.72401
Latency	20	.3264	.02325	.00520

One-Sample Statistics

Chord 2700	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0452	.00332	.00074
Bandwidth Consumption	20	51385.2255	829.14186	185.40176
Latency	20	.3316	.02658	.00594

One-Sample Statistics

Chord 3600	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0656	.00997	.00223
Bandwidth Consumption	20	50997.2382	755.06388	168.83742
Latency	20	.3364	.02325	.00520

One-Sample Statistics

Kad 900	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	2.9849	.04407	.00985
Bandwidth Consumption	20	39709.6265	674.47501	150.81720
Latency	20	.1969	.01356	.00303

One-Sample Statistics

Kad 1800	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	2.9869	.03475	.00777
Bandwidth Consumption	20	27669.8023	423.06498	94.60021
Latency	20	.1997	.01198	.00268

One-Sample Statistics

Kad 2700	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	2.9997	.01183	.00265
Bandwidth Consumption	20	24672.7762	114.42224	25.58559
Latency	20	.2003	.01376	.00308

One-Sample Statistics

Kad 3600	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	2.9989	.01258	.00281
Bandwidth Consumption	20	22428.7021	508.49383	113.70268
Latency	20	.1924	.01575	.00352

Delivery ratio from Chord

One-Sample Statistics

Chord 900	Number of replications	Mean	Std. Deviation	Std. Error Mean
Delivery ratio	20	.7243	.03566	.00797

One-Sample Statistics

Chord 1800	Number of replications	Mean	Std. Deviation	Std. Error Mean
Delivery ratio	20	.8566	.03591	.00803

One-Sample Statistics

Chord 2700	Number of replications	Mean	Std. Deviation	Std. Error Mean
Delivery ratio	20	.8937	.02665	.00596

One-Sample Statistics

Chord 3600	Number of replications	Mean	Std. Deviation	Std. Error Mean
Delivery ratio	20	.9200	.01933	.00432

One-Sample Statistics

Kademlia 900	Number of replications	Mean	Std. Deviation	Std. Error Mean
Delivery ratio	20	.9900	.00000	.00000

One-Sample Statistics

Kademlia 1800	Number of replications	Mean	Std. Deviation	Std. Error Mean
Delivery ratio	20	.9900	.00000	.00000

One-Sample Statistics

Kademlia 2700	Number of replications	Mean	Std. Deviation	Std. Error Mean
Delivery ratio	20	.9900	.00000	.00000

One-Sample Statistics

Kademlia 3600	Number of replications	Mean	Std. Deviation	Std. Error Mean
Delivery ratio	20	.9900	.00000	.00000

Results from Recursive routing

One-Sample Statistics

Chord 900	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	3.0575	.00633	.00142
Latency	20	.1305	.00010	.00002
Delivery ratio	20	.7964	.00139	.00031
Bandwidth Consumption	20	36693.5495	4.24150	.94843

One-Sample Statistics

Chord 1800	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	3.0660	.00000	.00000
Latency	20	.1331	.00008	.00002
Delivery ratio	20	.9075	.00171	.00038
Bandwidth Consumption	20	37331.2461	41.21929	9.21691

One-Sample Statistics

Chord 2700	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	3.0653	.00351	.00078
Latency	20	.1341	.00323	.00072
Delivery ratio	20	.9400	.00000	.00000
Bandwidth Consumption	20	37522.0053	28.32365	6.33336

One-Sample Statistics

Chord 3600	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	3.0844	.00326	.00073
Latency	20	.1300	.00000	.00000
Delivery ratio	20	.9600	.00000	.00000
Bandwidth Consumption	20	37600.8843	20.03071	4.47900

One-Sample Statistics

Kademlia 900	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.2307	.01252	.00280
Latency	20	.0525	.00127	.00028
Delivery ratio	20	.1443	.00355	.00079
Bandwidth Consumption	20	2250.7172	17.20557	3.84728

One-Sample Statistics

Kademlia 1800	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.3506	.00584	.00131
Latency	20	.0580	.00051	.00011
Delivery ratio	20	.1851	.00309	.00069
Bandwidth Consumption	45	2163.6791	66.16874	9.86385

One-Sample Statistics

Kademlia 2700	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.3856	.00277	.00062
Latency	20	.0540	.00000	.00000
Delivery ratio	20	.2293	.00634	.00142
Bandwidth Consumption	20	1743.8468	37.61226	8.41036

One-Sample Statistics

Kademlia 3600	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.4447	.00271	.00061
Latency	20	.0600	.00000	.00000
Delivery ratio	20	.2694	.00529	.00118
Bandwidth Consumption	20	3700.3307	26.20542	5.85971

Appendix A2

This section gives the numerical results for section 4.2.2.

One-Sample Statistics

Chord 900	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0345	.01195	.00267
Bandwidth Consumption	20	50648.4113	2319.32573	518.61700
Latency	20	.3364	.02325	.00520

One-Sample Statistics

Chord 1800	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0492	.00557	.00125
Bandwidth Consumption	20	51564.4327	1148.10467	256.72401
Latency	20	.3264	.02325	.00520

One-Sample Statistics

Chord 2700	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0452	.00332	.00074
Bandwidth Consumption	20	51385.2255	829.14186	185.40176
Latency	20	.3316	.02658	.00594

One-Sample Statistics

Chord 3600	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0656	.00997	.00223
Bandwidth Consumption	20	50997.2382	755.06388	168.83742
Latency	20	.3364	.02325	.00520

One-Sample Statistics

Kademlia 900	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	2.9849	.04407	.00985
Bandwidth Consumption	20	39709.6265	674.47501	150.81720
Latency	20	.1969	.01356	.00303

One-Sample Statistics

Kademlia 1800	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	2.9869	.03475	.00777
Bandwidth Consumption	20	27669.8023	423.06498	94.60021
Latency	20	.1997	.01198	.00268

One-Sample Statistics

Kademlia 2700	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	2.9997	.01183	.00265
Bandwidth Consumption	20	24672.7762	114.42224	25.58559
Latency	20	.2003	.01376	.00308

One-Sample Statistics

Kademlia 3600	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	2.9989	.01258	.00281
Bandwidth Consumption	20	22428.7021	508.49383	113.70268
Latency	20	.1924	.01575	.00352

One-Sample Statistics

GIA 900	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.3633	.29152	.06519
Bandwidth Consumption	20	16863.3180	1352.70896	302.47492
Latency	20	4.1208	.19322	.04320

One-Sample Statistics

GIA 1800	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.2600	.23431	.05239
Bandwidth Consumption	20	14894.9126	1250.16088	279.54447
Latency	20	18.8986	.77666	.17367

One-Sample Statistics

GIA 2700	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.3586	.17981	.04021
Bandwidth Consumption	20	16000.2402	1019.57203	227.98324
Latency	20	43.0147	2.06679	.46215

One-Sample Statistics

GIA 3600	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.1573	.21877	.04892
Bandwidth Consumption	20	13506.5836	769.84625	172.14286
Latency	20	59.0113	1.18756	.26555

Appendix A3

This section gives the numerical results for section 4.2.3.

One-Sample Statistics

Chord 900	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	5.5775	.00633	.00142
Latency	20	.4351	.00313	.00070
Bandwidth Consumption	20	58993.2758	81.12566	18.14025

One-Sample Statistics

Chord 1800	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0675	.00633	.00142
Latency	20	5.6600	.00000	.00000
Bandwidth Consumption	20	66181.9095	74.08940	16.56689

One-Sample Statistics

Chord 2700	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	5.6848	.00241	.00054
Latency	20	.4355	.00280	.00063
Bandwidth Consumption	20	68242.6158	87.15286	19.48797

One-Sample Statistics

Chord 3600	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	5.6942	.00337	.00075
Latency	20	.4300	.00000	.00000
Bandwidth Consumption	20	69319.1513	275.30406	61.55986

One-Sample Statistics

Kademlia 900	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	3.9254	.00270	.00060
Latency	20	.3154	.00290	.00065
Bandwidth Consumption	20	89521.2602	136.07002	30.42618

One-Sample Statistics

Kademlia 1800	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	3.8857	.00289	.00065
Latency	20	.3043	.00298	.00067
Bandwidth Consumption	20	60861.7313	327.59504	73.25248

One-Sample Statistics

Kademlia 2700	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	3.9398	.00527	.00118
Latency	20	.3100	.00000	.00000
Bandwidth Consumption	20	53844.0795	12.03090	2.69019

One-Sample Statistics

Kademlia 3600	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	3.9444	.00293	.00065
Latency	20	.3100	.00000	.00000
Bandwidth Consumption	20	50254.7966	338.27441	75.64046

One-Sample Statistics

GIA 900	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.6773	.00761	.00170
Latency	20	.8164	.03164	.00708
Bandwidth Consumption	20	21186.3685	44.31405	9.90892

One-Sample Statistics

GIA 1800	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.7559	.00330	.00074
Latency	20	6.0147	.10444	.02335
Bandwidth Consumption	20	21232.5876	13.55182	3.03028

One-Sample Statistics

GIA 2700	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.7100	.00000	.00000
Latency	20	15.5839	1.55056	.34672
Bandwidth Consumption	20	20770.0395	56.28897	12.58660

One-Sample Statistics

GIA 3600	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.7075	.00633	.00142
Latency	20	25.2297	2.02345	.45246
Bandwidth Consumption	20	20567.7326	141.96991	31.74544

Appendix A4

This section gives the numerical results for section 4.2.4.

One-Sample Statistics

Chord 15	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0627	.00694	.00155
Latency	20	.3047	.00257	.00057
Bandwidth consumption	20	65097.3873	28.35328	6.33999

hop count

One-Sample Statistics

Chord 30	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0675	.00633	.00142
Latency	20	.3051	.00313	.00070
Bandwidth consumption	20	54937.3858	4.30212	.96198

One-Sample Statistics

Chord 45	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0699	.00537	.00120
Latency	20	.2962	.00312	.00070
Bandwidth consumption	20	51611.0047	22.17657	4.95883

One-Sample Statistics

Chord 60	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	4.0656	.00997	.00223
Latency	20	.3364	.02325	.00520
Bandwidth consumption	20	50997.2382	755.06388	168.83742

One-Sample Statistics

Kademlia 15	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	3.0233	.00285	.00064
Latency	20	.1800	.00000	.00000
Bandwidth consumption	20	68128.4285	62.34808	13.94145

One-Sample Statistics

Kademlia 30	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	3.0254	.00270	.00060
Latency	20	.1754	.00290	.00065
Bandwidth consumption	20	38064.0609	8.13615	1.81930

One-Sample Statistics

Kademlia 45	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	2.9963	.02346	.00524
Latency	20	.1708	.00639	.00143
Bandwidth consumption	20	27244.2244	187.22726	41.86529

One-Sample Statistics

Kademlia 60	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	2.9989	.01258	.00281
Latency	20	.1924	.01575	.00352
Bandwidth consumption	20	22428.7021	508.49383	113.70268

One-Sample Statistics

GIA 15	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.5948	.08536	.01909
Latency	20	113.3926	5.62532	1.25786
Bandwidth consumption	20	27668.6751	938.76778	209.91486

One-Sample Statistics

GIA 30	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.3010	.18542	.04146
Latency	20	76.6305	6.12690	1.37002
Bandwidth consumption	20	26088.2633	1750.71198	391.47110

One-Sample Statistics

GIA 45	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.4047	.20295	.04538
Latency	20	54.2239	2.44300	.54627
Bandwidth consumption	20	20314.5181	975.41024	218.10836

One-Sample Statistics

GIA 60	Number of replications	Mean	Std. Deviation	Std. Error Mean
Hop count	20	1.1573	.21877	.04892
Latency	20	59.0113	1.18756	.26555
Bandwidth consumption	20	13506.5836	769.84625	172.14286

Appendix B

Simulation Parameters

Appendix B1

This section shows the values of simulation parameters used in the simulation script *default.ini*.

```
# --- Test applicaiton settings ---
# KBRTestApp settings
**.tier1*.kbrTestApp.testMsgInterval = 60s
**.tier1*.kbrTestApp.msgHandleBufSize = 8
**.tier1*.kbrTestApp.lookupNodeIds = true

# GIASearchApp settings
**.tier1*.giaSearchApp.messageDelay = 60s
**.tier1*.giaSearchApp.randomNodes = true
**.tier1*.giaSearchApp.randomKeys = 20
**.tier1*.giaSearchApp.maximumKeys = 100
**.tier1*.giaSearchApp.minKeyProbability = 0.1
**.tier1*.giaSearchApp.maxKeyProbability = 0.15
**.tier1*.giaSearchApp.maxResponses = 10
**.tier1*.giaSearchApp.routeMessages = true
**.tier1*.giaSearchApp.searchMessages = false



# --- Overlay settings ---

# Chord settings
**.overlay*.chord.joinRetry = 2
**.overlay*.chord.joinDelay = 10s
**.overlay*.chord.stabilizeRetry = 1
**.overlay*.chord.stabilizeDelay = 20s
**.overlay*.chord.fixfingersDelay = 120s
**.overlay*.chord.checkPredecessorDelay = 5s
**.overlay*.chord.routingType = "iterative"
**.overlay*.chord.successorListSize = 8
**.overlay*.chord.aggressiveJoinMode = true
**.overlay*.chord.extendedFingerTable = false
**.overlay*.chord.numFingerCandidates = 3
**.overlay*.chord.proximityRouting = false
**.overlay*.chord.memorizeFailedSuccessor = false
```

```
**.overlay*.chord.mergeOptimizationL1 = false
**.overlay*.chord.mergeOptimizationL2 = false
**.overlay*.chord.mergeOptimizationL3 = false
**.overlay*.chord.mergeOptimizationL4 = false

# kademlia settings
**.overlay*.kademlia.lookupRedundantNodes = 8
**.overlay*.kademlia.lookupParallelPaths = 1
**.overlay*.kademlia.lookupParallelRpcs = 3
**.overlay*.kademlia.lookupMerge = true
**.overlay*.kademlia.routingType = "iterative"
**.overlay*.kademlia.minSiblingTableRefreshInterval = 1000s
**.overlay*.kademlia.minBucketRefreshInterval = 1000s
**.overlay*.kademlia.maxStaleCount = 0
**.overlay*.kademlia.k = 8
**.overlay*.kademlia.s = 8
**.overlay*.kademlia.b = 1
**.overlay*.kademlia.pingNewSiblings = true

# gia settings
**.overlay*.gia.maxNeighbors = 50
**.overlay*.gia.minNeighbors = 10
**.overlay*.gia.maxTopAdaptionInterval = 120s
**.overlay*.gia.topAdaptionAggressiveness = 256
**.overlay*.gia.maxLevelOfSatisfaction = 1.00
**.overlay*.gia.updateDelay = 60s
**.overlay*.gia.maxHopCount = 10
**.overlay*.gia.messageTimeout = 180s
**.overlay*.gia.sendTokenTimeout = 5s
**.overlay*.gia.neighborTimeout = 250s
**.overlay*.gia.tokenWaitTime = 5s
**.overlay*.gia.keyListDelay = 100s
**.overlay*.gia.outputNodeDetails = false
**.overlay*.gia.optimizeReversePath = false

# Generic settings
**.overlay*.*.debugOutput = true
**.overlay*.*.hopCountMax = 50
**.overlay*.*.recNumRedundantNodes = 3
**.overlay*.*.collectPerHopDelay = false
InetUnderlayNetwork.*.overlay*.*.drawOverlayTopology = false
SingleHostUnderlayNetwork.*.overlay*.*.drawOverlayTopology = false
**.overlay*.*.drawOverlayTopology = true
**.overlay*.*.useCommonAPIforward = false
**.overlay*.*.routingType   =   "iterative"     #"iterative   exhaustive-
iterative semi-recursive full-recursive source-routing-recursive"
**.overlay*.*.keyLength = 160
**.overlay*.*.joinOnApplicationRequest = false
**.overlay.*.localPort = 1024

# general overlay lookup settings
**.overlay*.*.lookupRedundantNodes = 1
**.overlay*.*.lookupParallelPaths = 1
**.overlay*.*.lookupParallelRpcs = 1
**.overlay*.*.lookupSecure = false
**.overlay*.*.lookupMerge = false
**.overlay*.*.lookupUseAllParallelResponses = false
```

```
**.overlay*.*.lookupStrictParallelRpcs = false
**.overlay*.*.lookupNewRpcOnEveryTimeout = false
**.overlay*.*.lookupNewRpcOnEveryResponse = false
**.overlay*.*.lookupFinishOnFirstUnchanged = false
**.overlay*.*.lookupFailedNodeRpcs = false
**.overlay*.*.routeMsgAcks = false

# bootstrapList configuration
**.bootstrapList.debugOutput = true
**.bootstrapList.mergeOverlayPartitions = false
**.bootstrapList.maintainList = false

# neighbor cache settings
**.neighborCache.enableNeighborCache = false
**.neighborCache.rttExpirationTime = 100s
**.neighborCache.maxSize = 50

# ---- BaseRpc settings ----

**.rpcUdpTimeout = 1.5s
**.rpcKeyTimeout = 10.0s
**.rpcExponentialBackoff = false

# ---- UnderlayConfigurator settings ----

# UnderlayConfigurator module settings
*.underlayConfigurator.transitionTime = 0s
*.underlayConfigurator.measurementTime = -1s
*.underlayConfigurator.gracefulLeaveDelay = 15s
*.underlayConfigurator.gracefulLeaveProbability = 0.5
# disable churn for SingleHost networks
SingleHostUnderlayNetwork.chunderlayConfigurator.churnGeneratorTypes   =
""
#  any  combination  of  "NoChurn",  "LifetimeChurn",  "ParetoChurn"  and
"RandomChurn" separated by spaces
*.underlayConfigurator.churnGeneratorTypes                              =
"oversim.common.ParetoChurn"

# ChurnGenerator configuration
*.churnGenerator*.initPhaseCreationInterval = 1s
*.churnGenerator*.targetOverlayTerminalNum = 10
*.churnGenerator*.lifetimeMean = 3600.0s
*.churnGenerator*.deadtimeMean = 3600.0s
*.churnGenerator*.lifetimeDistName = "weibull"
*.churnGenerator*.lifetimeDistPar1 = 5.0
*.churnGenerator*.noChurnThreshold = 0s
# RandomChurn (obsolete)
*.churnGenerator*.targetMobilityDelay = 300s
*.churnGenerator*.targetMobilityDelay2 = 20s
*.churnGenerator*.churnChangeInterval = 0s
*.churnGenerator*.creationProbability = 0.5i
*.churnGenerator*.migrationProbability = 0.0key
*.churnGenerator*.removalProbability = 0.5

# use globalFunctions?
*.globalObserver.globalFunctionsType = ""
*.globalObserver.useGlobalFunctions = false
```

```
# global statistics
*.globalObserver.globalStatistics.outputMinMax = false
*.globalObserver.globalStatistics.outputStdDev = false
*.globalObserver.globalStatistics.globalStatTimerInterval = 0s
*.globalObserver.globalStatistics.measureNetwInitPhase = false

# GlobalNodeList settings
*.globalObserver.globalNodeList.maxNumberOfKeys = 100
*.globalObserver.globalNodeList.keyProbability = 0.1
*.globalObserver.globalNodeList.maliciousNodeProbability = 0.0
*.globalObserver.globalNodeList.maliciousNodeChange = false
*.globalObserver.globalNodeList.maliciousNodeChangeStartTime = 200s
*.globalObserver.globalNodeList.maliciousNodeChangeRate = 0.05
*.globalObserver.globalNodeList.maliciousNodeChangeInterval = 100s
*.globalObserver.globalNodeList.maliciousNodeChangeStartValue = 0
*.globalObserver.globalNodeList.maliciousNodeChangeStopValue = 0.5

# GlobalObserver configuration
*.globalObserver.globalTraceManager.traceFile = ""
*.globalObserver.globalParameters.printStateToStdOut = false
*.globalObserver.globalParameters.topologyAdaptation = false


# InetUnderlayNetwork configuration
InetUnderlayNetwork.outRouter*.outDeviceType                                =
"oversim.underlay.singlehostunderlay.TunOutDevice"
**.mtu = 65000
**.parser = "oversim.common.GenericPacketParser"
**.appParser = "oversim.common.GenericPacketParser"
**.gatewayIP = ""

# InetUnderlay backbone configuration
InetUnderlayNetwork.underlayConfigurator.terminalTypes                      =
"oversim.underlay.inetunderlay.InetOverlayHost"
InetUnderlayNetwork.**.channelTypes  =  "oversim.common.inet_ethernetline
oversim.common.inet_dsl"
InetUnderlayNetwork.backboneRouterNum = 1
InetUnderlayNetwork.overlayBackboneRouterNum = 0
InetUnderlayNetwork.accessRouterNum = 2
InetUnderlayNetwork.overlayAccessRouterNum = 0
InetUnderlayNetwork.connectivity = 0.8
InetUnderlayNetwork.underlayConfigurator.startIP = "1.1.0.1"
InetUnderlayNetwork.outRouterNum = 0

# default overlay and application
#    Here    **    includes    *.globalObserver.globalTraceManager    and
*.churnGenerator*
**.overlayType = "oversim.overlay.chord.ChordModules"
**.tier1Type = "oversim.applications.kbrtestapp.KBRTestAppModules"
**.tier2Type = "oversim.common.TierDummy"
**.tier3Type = "oversim.common.TierDummy"
**.numTiers = 1
```

Appendix B2

This section shows a proportion of the simulation script *omnetpp.ini*.

```
[Config Chord100_1]
description = Chord (iterative, InetUnderlayNetwork)
network = oversim.underlay.inetunderlay.InetUnderlayNetwork
*.underlayConfigurator.churnGeneratorTypes                                    =
"oversim.common.ParetoChurn"
**.overlayType = "oversim.overlay.chord.ChordModules"
**.tier1Type = "oversim.applications.kbrtestapp.KBRTestAppModules"
InetUnderlayNetwork.backboneRouterNum = 1
InetUnderlayNetwork.overlayAccessRouterNum = 2
**.targetOverlayTerminalNum = 100
seed-set = 21000

[Config Chord1000_1]
description = Chord (iterative, InetUnderlayNetwork)
network = oversim.underlay.inetunderlay.InetUnderlayNetwork
*.underlayConfigurator.churnGeneratorTypes                                    =
"oversim.common.ParetoChurn"
**.overlayType = "oversim.overlay.chord.ChordModules"
**.tier1Type = "oversim.applications.kbrtestapp.KBRTestAppModules"
InetUnderlayNetwork.backboneRouterNum = 1
InetUnderlayNetwork.overlayAccessRouterNum = 2
**.targetOverlayTerminalNum = 1000
seed-set = 21000



[Config Kad100_1]
description = Kademlia (iterative, InetUnderlayNetwork)
network = oversim.underlay.inetunderlay.InetUnderlayNetwork
*.underlayConfigurator.churnGeneratorTypes                                    =
"oversim.common.ParetoChurn"
**.overlayType = "oversim.overlay.kademlia.KademliaModules"
**.tier1Type = "oversim.applications.kbrtestapp.KBRTestAppModules"
**.targetOverlayTerminalNum = 100
InetUnderlayNetwork.backboneRouterNum = 1
InetUnderlayNetwork.overlayAccessRouterNum = 2
seed-set = 21000

[Config Kad1000_1]
description = Kademlia (iterative, InetUnderlayNetwork)
network = oversim.underlay.inetunderlay.InetUnderlayNetwork
*.underlayConfigurator.churnGeneratorTypes                                    =
"oversim.common.ParetoChurn"
**.overlayType = "oversim.overlay.kademlia.KademliaModules"
**.tier1Type = "oversim.applications.kbrtestapp.KBRTestAppModules"
**.targetOverlayTerminalNum = 1000
InetUnderlayNetwork.backboneRouterNum = 1
InetUnderlayNetwork.overlayAccessRouterNum = 2
seed-set = 25000

[Config Gia100_1]
```

```
description = Gia (iterative, InetUnderlayNetwork)
network = oversim.underlay.inetunderlay.InetUnderlayNetwork
*.underlayConfigurator.churnGeneratorTypes                                =
"oversim.common.ParetoChurn"
**.overlayType = "oversim.overlay.gia.GiaModules"
**.tier1Type = "oversim.applications.giasearchapp.GIASearchAppModules"
**.targetOverlayTerminalNum = 100
InetUnderlayNetwork.backboneRouterNum = 1
InetUnderlayNetwork.overlayAccessRouterNum = 2
seed-set = 21000

[Config Gia1000_1]
description = Gia (iterative, InetUnderlayNetwork)
network = oversim.underlay.inetunderlay.InetUnderlayNetwork
*.underlayConfigurator.churnGeneratorTypes                                =
"oversim.common.ParetoChurn"
**.overlayType = "oversim.overlay.gia.GiaModules"
**.tier1Type = "oversim.applications.giasearchapp.GIASearchAppModules"
**.targetOverlayTerminalNum = 1000
InetUnderlayNetwork.backboneRouterNum = 1
InetUnderlayNetwork.overlayAccessRouterNum = 2

seed-set = 25000
```

Appendix C

Survey of P2P Simulator

This section shows the report on the P2P simulators we have surveyed.

Appendix C1 ns-2

Simulation architecture: ns-2 [3] is a popular non-commercial network simulator developed and used by global academic. It supports TCP, routing and multicast protocols over wired and wireless networks. ns-2 is an event-based simulator that simulates packet-level simulations. The simulator itself is written in C++ and simulation scripts are written in OTcl. The scripts would define the network environment and behaviour. Protocols and other network properties are composed as modules which are integrated together. Different ns-2 modules have been developed by individuals to accomplish their objectives, but the integration and usage of the modules are not often straightforward for other users who might have a different version of the network simulator. The simulator supports the churn behaviour of nodes joining and leaving P2P systems. It also supports network trace files from topology generators such as Brite [11].

In Figure A.1, we can see that the ns-2 simulator is composed of both C++ and OTcl. The simulator and functionalities itself are defined in C++, and OTcl scripts are used to describe and execute the experiments. Scripts defined in OTcl will invoke functions in the simulator.

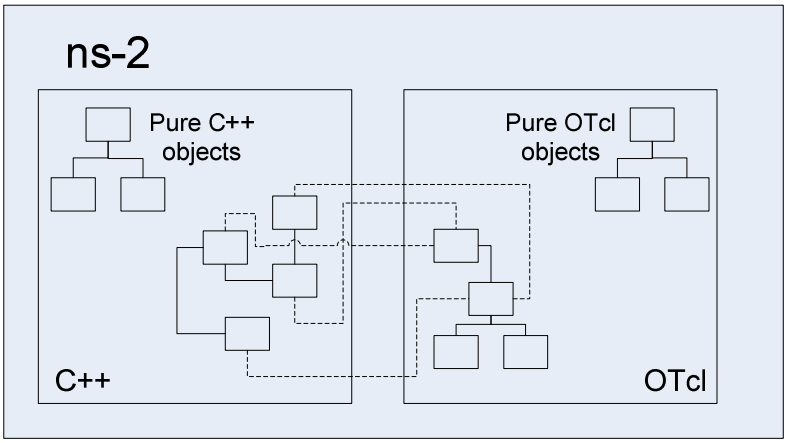


Figure A.1 A graphical representation of the ns-2 architecture. An object created with OTcl will have a corresponding object in C++.

	Existing core ns-2 capability	ns-2 contributed code
Applications	ping, vat, telnet, FTP, multicast FTP, HTTP, probabilistic and trace-driven traffic generators, webcache	NSWEB, Video traffic generator, MPEG generator, BonnTraffic, ProtoLib, AgentJ, SIP, NSIS, ns2voip, Agent/Plant
Transport layer	TCP (many variants), UDP, SCTP, XCP, TFRC, RAP, RTP Multicast: PGM, SRM, RLM, PLM	TCP PEP, SCPS-TP SNACK, TCP Pacing, DCCP, Simulation Cradle, TCP Westwood, SIMD, TCP-RH, MFTP, OTERS, TCP Eifel
Network layer	Unicast: IP, MobileIP, generic dist. vector and link state, IPinIP, source routing, Nixvector Multicast: SRM, generic centralized MANET: AODV, DSR, DSDV, TORA, IMEP	AODV+, AODV-UU, AOMDV, ns-click, ZRP, IS-IS, CDS, Dynamic Linkstate, DYMO, OLSR, ATM, AntNet, Mobile IPv6, IP micro-mobility, MobileIP, GPSR, RSVP, PGM, PLM, SSM, PUMA, ActiveNetworks
Link layer	ARP, HDLC, GAF, MPLS, LDP, Diffserv Queueing: DropTail, RED, RIO, WFQ, SRR, Semantic Packet Queue, REM, Priority, VQ MACs: CSMA, 802.11b, 802.15.4 (WPAN), satellite Aloha	802.16, 802.11e HCCA, 802.11e EDCA, 802.11a multirate, UWB DCC-MAC, TDMA DAMA, EURANE, UMTS, GPRS, BlueTooth, 802.11 PCF,, 802.11 PSM, MPLS, WFQ schedulers, Bandwidth Broker, CSFQ, BLUE
Physical layer	TwoWay, Shadowing, OmniAntennas, EnergyModel, Satellite Repeater	ET/SNRT/BER-based Phy, IR-UWB
Support	Random number generators, tracing, monitors, mathematical support, test suite, animation (nam), error models	Emulation, CANU mobility, BonnMotion mobility, SGB Topology Generators, NSG2, simd, ns2measure, ns-2/akaroa-2, yavista, tracegraph, huginn, multistate error model, RPI graphing package, jTrana, GEA,

Figure A.2 An example of ns-2 existing and contributed modules.

Usability: A manual on the usage of OTcl scripts on creating and describing ns-2 simulations can be found on its homepage [3]. In the manual, there are references to the simulator APIs, but the focus has been placed on demonstrating the usage of the OTcl scripting language. There is not sufficient documentation on the simulator API itself. Users need to study the modules themselves in detail and rely on the developers'

comments throughout the program. Online support through forums and mailing lists is still available, where users can seek for assistance, but responses are not guaranteed. From Figure A.2, we can see the structure of available ns-2 modules from both their developers and also the community. ns-2 does not support a graphical user interface (GUI). However, it allows a network animator, nam, to provide network animations. nam can be used to rearrange network graphs to assist with designing and debugging of simulations. In Figure A.3, we can see the graphical representation of a ns-2 simulation. The GUI enable users with easier access to customizing the simulation.

Scalability: Simulations of P2P systems in ns-2 are usually limited to hundreds of nodes at best. It is not as scalable as other simulators because it does not abstract the protocols of lower layers.

Statistics: At the end of a simulation run, ns-2 outputs trace files that contain packet level data. The data represent raw results that will need post processing for obtaining final results and for assessing their statistical accuracy.

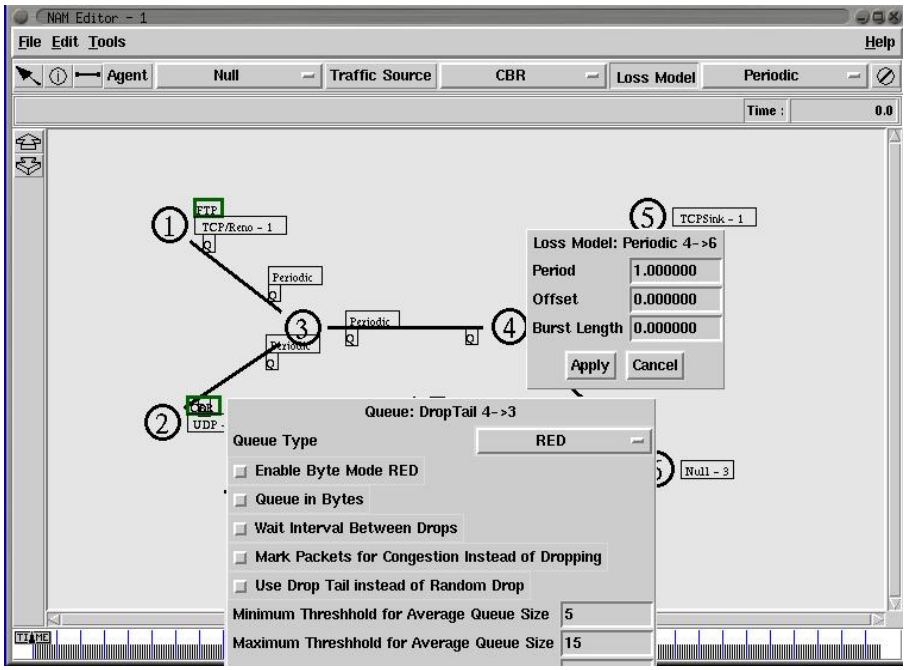


Figure A.3 The nam editor

Underlying network: ns-2 as a packet level simulator is able to simulate different network elements. They can be customized in simulation scripts in OTcl, for example, routers, link bandwidth, and latencies.

A new version of ns, ns-3, is still under development. New functionalities and new overlay protocols are added to new releases. Few P2P systems have been ported to the new simulator but no substantial P2P experiments are performed with ns-3 yet.

Appendix C2 PeerSim

Simulation architecture: PeerSim [4] is developed with Java. The architecture composes two simulation engines, which are cycle and event based. The cycle-based engine supports a large scale network simulation by abstracting details in the transport layer. The event-based engine can support more realistic simulations, but cannot support larger networks. New protocols can be added into the simulator, but not all code can be used between the two simulation engines simultaneously. PeerSim supports simulation of the churn behaviour of nodes joining and leaving.

Usability: Users can refer to the online documentation compiled by javadoc, and tutorials are available to learn about the two engines. However, the tutorials have only presented simple simulations, and users still have to read through the API to get a better understanding. PeerSim does not have a graphical user interface, and there is no support for visualisation during simulation. A list of contributed code from other users is maintained.

Scalability: The cycle-based engine can simulate up to 10^6 of peer nodes. This ability is achieved by abstracting the details in the transport layer, which means a loss of accuracy. The event-based engine cannot support as large networks but it is more accurate and realistic, compared to cycle- based simulations.

Statistics: At the end of a simulation run, PeerSim outputs files with recorded data predefined before simulation. The data represent raw results that will need post processing to proceed on with further analysis

Underlying network: The cycle-based engine abstracts details of the transport layer to improve scalability. The event based engine supports simulation details in the transport layer.

Appendix C3 OverSim

Simulation architecture: OverSim is an overlay network simulation framework developed in C++. Looking at Figure A.4, Oversim is split into three levels: application, overlay and underlay. The application level defines test applications such as key based routing and DHT services. Key Based Routing (KBR) is a lookup method used to find the closest host from a request. Routing can be in either the iterative or recursive mode in KBR. The overlay level defines different structured and un-structured P2P protocols. The underlay level supports a range of the transport, IP and network layer. In simple networks, the framework adopts trace files from the CAIDA/Skitter project [12] to support simulation of large scale networks. The INET networks support simulation of the network layer and data link layer. Single Host networks act as middleware to support OverSim on a real network. Users can implement their own overlay protocol in OverSim. The simulator also supports simulation of churn behaviour. Furthermore, churn behaviour can modelled by using such probability distributions as Weibull and Pareto.

Usability: On OverSim's homepage, users have access to both the module and C++ API documentation. There are also guides on how to use OverSim, information on how to implement new overlay protocols, and extra guides on some parts of OverSim. There is also a discussion forum on Google Groups where developers would assist users. To operate OverSim, there are two configuration files to modify, where users can customize their own simulations. The configuration files are written with OMNeT++ ini file syntax,

and they are rather straightforward. Figure A.5 displays an example of OverSim's GUI, and virtualisations of a simulation. The top right corner represents the modelled network, and below there is an update of a particular parameter during simulation. The other half of the diagram displays the simulation controls interface and the current status of the simulation.

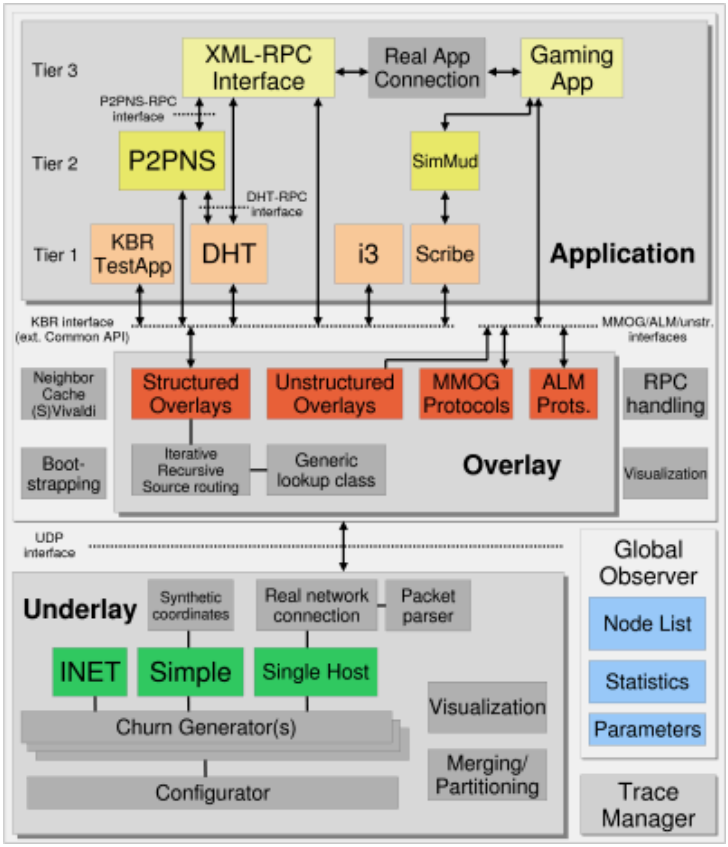


Figure A.4 Oversim's architecture

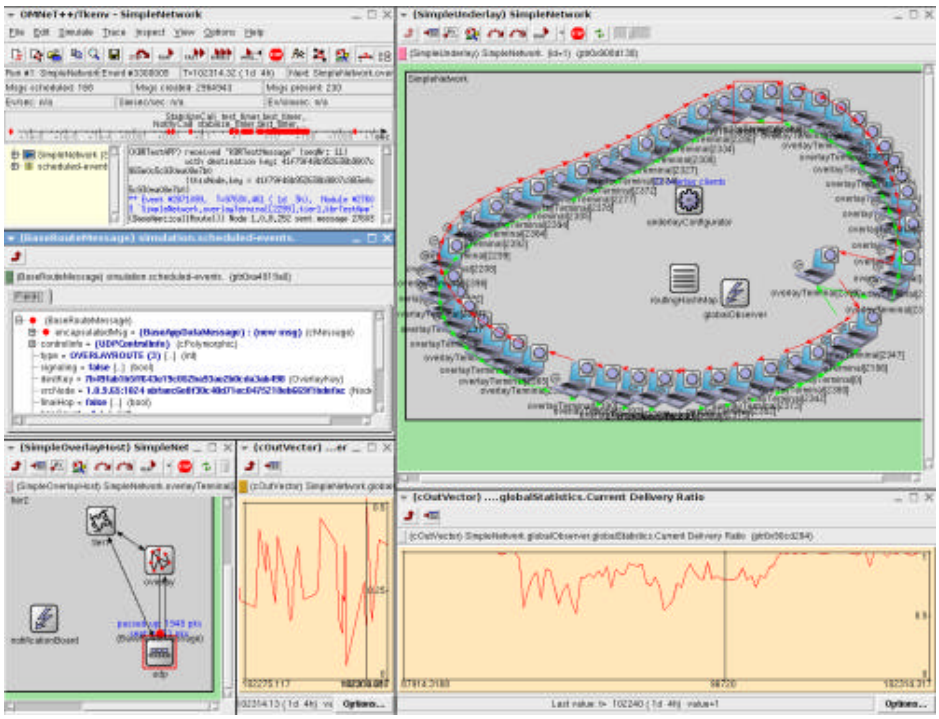


Figure A.5 Oversim GUI and visualization of a simulation.

Scalability: Using the simple network with abstracted details in the underlay, the simulated network can scale up to 100, 000 nodes. INET underlays are less scalable because they simulate network properties, and network typically scales up to thousands of nodes. In OverSim, churn rates also play a part in simulation performance. A more dynamic network with higher churn rates would result in a less scalable network.

Statistics: OverSim collects various statistical data such as hop count, latencies, number of sent and received messages and success rate. Later versions of OverSim, version 20090908 and beyond have included Python scripts to assist users to post-process the data and generate gnuplot graphs.

Underlying network: OverSim underlay have the flexibility to choose between three network properties. The simple network uses trace files from the CAIDA/Skitter project which represent measurements from the real Internet. The INET framework allows users to define various network properties such as bandwidth, latencies and routers. The Single

Host component acts as a middleware to support running OverSim on a real network such as PlanetLab.

Appendix C4 Neurogrid

Simulation architecture: The Neurogrid simulator [5] is another message-level simulator developed in Java. The simulator has a number of flexible P2P protocol extensions; with Freenet, Gnutella and NeuroGrid as the built-in protocols. Neurogrid has a few abstracted classes that are shared across different P2P networks. Upon simulation, the P2P network extends the generic classes to better simulate the protocol behaviour. Neurogird is a message level simulator, and lower network layers are abstracted. There are several network topologies that are supported, Ring for example. Churn behaviour is not supported.

Usability: There is extensive documentation on the Neurogrid's homepage. Guides and tutorial are available to assist beginners. Neurogird can be executed from the command line or a GUI to customize simulations. Virtualisation of simulations is available from the GUI. Users have access to the mailing support list, but the development of the simulator has been stopped after 2003. There is no documentation regarding the API. In Figure A.6, on the left, is the visualization of a Ring network, and on the right are the simulation parameters.

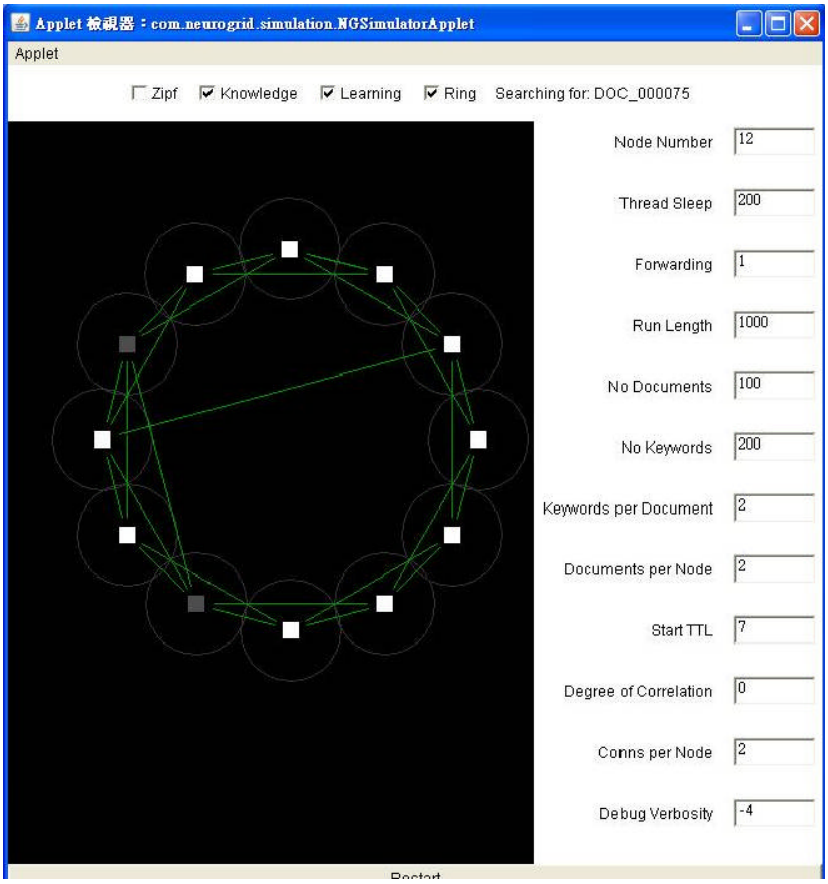


Figure A.6 Neurogrid's GUI, and visualization of a simulation run

Scalability: Neurogrid supports simulation of up to 300,000 nodes, but the stability of large network simulations has not been verified.

Statistics: Observations of predefined parameters are collected for output. Users can define their own methods to collect other data.

Underlying network: Neurogrid does not have support for simulating lower level network behaviour.

Appendix C5 PlanetSim

Simulation architecture: PlanetSim [24] is an event based simulator developed with Java. Its architecture has three layers: network, overlay, and application. The network layer allows users to implement their own network model or use the predefined network structures such as ring, circular or random. The overlay consists of models of P2P protocols and node behaviour, to assist the key based routing infrastructure used in the upper layer. The application layer provides the DHT services that define the routing and processing of messages in applications. Simulation scripts are written in Java. The scripts would define the network environment, including network properties such as latencies and bandwidth. The script could also define events to simulate churn. The network layer supports very limited details of the underlying networks. It only defines the network layout between a ring network and a circular network.

Usability: There exists extensive support for users on the PlanetSim website [13], ranging from the installation guide, tutorials, power point slides, publication papers, and javadoc documentation. PlanetSim does not provide a GUI or visualisation during simulation.

Scalability: PlanetSim can support up to 100,000 nodes. Large scale simulation is achieved by abstracting details in the underlay network.

Statistics: PlanetSim does not perform data collection or produce trace file. It is up to each user to implement their own methods to collect output data.

Underlying network: The network module only defines the network layout between a ring network and a circular network, and abstracts all other details in the underlay network. The simulator does support network trace files from GT-ITM, and other topology generators such as Brite.

Appendix C6 P2PSim

Simulation architecture: P2PSim [6] is another message level simulator, developed in C++. Simulations in P2PSim require three components: topology, protocol, and events which need to be loaded upon execution. The current inbuilt protocols are mainly structured, Chord, Kademlia, Accordion, Koorde, Kelips, and Tapestry. Churn behaviour is supported.

Usability: Users can find documentation on P2PSim's homepage [6], with brief setup instructions, short examples of commands and outputs. There is also a brief explanation of the output parameters. There is no GUI to customize simulations or simulation visualization. There is no documentation on the API, and only simple graph representation of the relationships between C++ classes is given. There is very little support to help users in extending other P2P protocols in P2PSim.

Scalability: P2PSim support simulations with up to 3000 nodes, but the stability of large network simulations has not been verified.

Statistics: Output data predefined parameters are available at the end of a simulation, and users need to define their own methods for collecting other data if necessary.
Underlying network – P2PSim does have some support for simulating lower level network properties.