# STRUCTURED PEER-TO-PEER OVERLAYS FOR NATED CHURN INTENSIVE NETWORKS

FARIDA CHOWDHURY


Degree of Doctor of Philosophy

Division of Computing Science and Mathematics
University of Stirling

December 2015

## DECLARATION

I, Farida Chowdhury, hereby declare that this work has not been submitted for any other degree at this University or any other institution and that, except where reference is made to the work of other authors, the material presented is original.

*Stirling, December 2015*

Farida Chowdhury

## ABSTRACT

The wide-spread coverage and ubiquitous presence of mobile networks has propelled the usage and adoption of mobile phones to an unprecedented level around the globe. The computing capabilities of these mobile phones have improved considerably, supporting a vast range of third party applications. Simultaneously, Peer-to-Peer (P2P) overlay networks have experienced a tremendous growth in terms of usage as well as popularity in recent years particularly in fixed wired networks. In particular, Distributed Hash Table (DHT) based Structured P2P overlay networks offer major advantages to users of mobile devices and networks such as scalable, fault tolerant and self-managing infrastructure which does not exhibit single points of failure. Integrating P2P overlays on the mobile network seems a logical progression; considering the popularities of both technologies. However, it imposes several challenges that need to be handled, such as the limited hardware capabilities of mobile phones and churn (i.e. the frequent join and leave of nodes within a network) intensive mobile networks offering limited yet expensive bandwidth availability. This thesis investigates the feasibility of extending P2P to mobile networks so that users can take advantage of both these technologies: P2P and mobile networks.

This thesis utilises OverSim, a P2P simulator, to experiment with the performance of various P2P overlays, considering high churn and bandwidth consumption which are the two most crucial constraints of mobile networks. The experiment results show that Kademlia and EpiChord are the two most appropriate P2P overlays that could be implemented in mobile networks. Furthermore, Network Address Translation (NAT) is a major barrier to the adoption of P2P overlays in mobile networks. Integrating NAT traversal approaches with P2P overlays is a crucial step for P2P overlays to operate successfully on mobile networks. This thesis presents a general approach of NAT traversal for ring based overlays without the use of a single dedicated server which is then implemented in OverSim. Several experiments have been performed under NATs to determine the suitability of the chosen P2P overlays under NATed environments. The results show that the performance of these overlays is comparable in terms of successful lookups in both NATed and non-NATed environments; with Kademlia and EpiChord exhibiting the best performance.

The presence of NATs and also the level of churn in a network influence the routing techniques used in P2P overlays. Recursive routing is more resilient to IP connectivity restrictions posed by NATs but not very robust in high churn environments, whereas iterative routing is more suitable to high churn networks, but difficult to use in NATed environments. Kademlia

supports both these routing schemes whereas EpiChord only supports the iterating routing. This undermines the usefulness of EpiChord in NATed environments.

In order to harness the advantages of both routing schemes, this thesis presents an adaptive routing scheme, called Churn Aware Routing Protocol (ChARP), combining recursive and iterative lookups where nodes can switch between recursive and iterative routing depending on their lifetimes. The proposed approach has been implemented in OverSim and several experiments have been carried out. The experiment results indicate an improved performance which in turn validates the applicability and suitability of ChARP in NATed environments.

# ACKNOWLEDGMENTS

This thesis would not have been possible without the support of a number of people, all of whom I wish to acknowledge with gratitude for their help and support.

First of all, I would like to thank my supervisor Dr. Mario Kolberg, an extraordinary mentor, for his continuous guidance, inspiration, insightful comments and support throughout this research. He sparked my interests to conduct innovative research and challenged me to reach my very best. I never felt that I had to wait for my research work's feedback as he always provided quick and constructive feedback. I am deeply grateful for his patience.

I would like to thank my second supervisor Professor Evan Magill for his valuable insights and advice.

This work was funded by the Scottish Informatics and Computer Science Alliance (SICSA) to whom thanks are due.

My heartfelt thanks go to everyone in the School of Computing Science for making it a delightful place to work and for all the support I received during my PhD. Special thanks to my colleagues and friends Erin, Dalila, Jamie and Claire for supporting me throughout the years.

I also would like to express my sincere thanks to Dr. Andrew Abel for proof-reading this thesis within a very short period of time.

Special thanks to Shifa and Niaz for sharing my joys and the hardships during my PhD years.

On the personal level, I would like to deeply thank my parents, Muraduzzaman Chowdhury and Gulshana Zaman, for their help, encouragement and unwavering belief in me. Without their support and prayers I wouldn't come that far. Special thanks to my twin sister Fahmida Chowdhury for providing me encouragement and mental support over phone from thousands of miles away. Special mention to my mother-in-law, Sultana Ferdous, who never forgets to keep me in her prayers.

Finally, my greatest thanks go to my beloved husband Md. Sadek Ferdous Ripul, for keeping me sane and support me all the time. Without his encouragement and support it would not have been possible to finish my thesis.

*To Abbu and Amma, my respected parents*

*To Ripul, my beloved husband*

*And to Apa, Leena, Mahruba and Aquib, my lovely sisters and brother.*

## LIST OF PUBLICATIONS

- Farida Chowdhury, Jamie Furness and Mario Kolberg. "Performance Analysis of Structured Peer-to-Peer Overlays". A journal paper submission in progress.
- Jamie Furness, Farida Chowdhury and Mario Kolberg. "An Evaluation of EpiChord in OverSim". In *Proceedings of the 5th International Conference on Networks and Communication (NETCOM 2013)*, Chennai, India, 27 – 29 December, 2013.
- Farida Chowdhury and Mario Kolberg. "Performance Evaluation of Structured Peer-to-Peer Overlays for use on Mobile Networks". In *Proceedings of the 6th International Conference on Developments in eSystems Engineering (DeSe 2013)*, Dubai, UAE, 16 – 18 December, 2013.
- Farida Chowdhury and Mario Kolberg. "Performance Evaluation of EpiChord under High Churn". In *Proceedings of the 8th ACM Performance Monitoring, Measurement and evaluation of Heterogeneous Wireless and Wired Networks Workshop (PM2HW2N 2013)*, Barcelona, Spain, 3 – 8 November, 2013.
- Farida Chowdhury and Mario Kolberg. "A Survey of Peer-to-Peer Solutions in Mobile and Cellular Networks". In *Proceedings of the 13th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNet 2012)*, Liverpool, UK, 25 – 26 June, 2012.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ACRONYMS

| | |
|---|---|
| API | Application Programming Interface |
| CAN | Content Addressable Network |
| CDN | Content Delivery Network |
| DES | Discrete Event Simulation |
| DHT | Distributed Hash Table |
| DML | Domain Modelling Language |
| ICMP | Internet Control Message Protocol |
| IP | Internet Protocol |
| KBR | Key-Based Routing |
| NAT | Network Address Translation |
| NED | Network Descriptor |
| P2P | Peer to Peer |
| PDA | Personal Digital Assistant |
| PNS | Proximity Neighbour Selection |
| RUDP | Reliable User Datagram Protocol |
| RPC | Remote Procedure Calls |
| SCTP | Stream Control Transmission Protocol |
| SSF | Scalable Simulation Framework |
| STUN | Simple Traversal of UDP through NAT |
| STUNT | Simple Traversal of UDP through NATs and TCP too |
| TCP | Transmission Control Protocol |
| TURN | Traversal Using Relays around NAT |
| TTL | Time-To-Live |
| UDP | User Datagram Protocol |
| UMTS | Universal Mobile Telecommunications Service |

Part I

# INTRODUCTION

# 1

# INTRODUCTION

## 1.1 MOTIVATION

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

P2P applications have been very successful on fixed, wired networks. With the explosive increase in popularity of smartphones, the consumption of data services on the move has increased substantially, a move which makes the availability of P2P overlay networks very desirable. The recent Ericsson Mobility Report suggests that by 2020, nearly 70% of the world's population will be using smartphones and about 90% of all the global population will be covered by mobile broadband networks. Also 80% of mobile data traffic will be generated from smartphones by 2020 [24].

Not surprisingly, the inherent properties of such mobile environments impose several significant challenges to deploying a P2P system on such networks. For example, mobile devices used in the mobile network often have strict bandwidth limits which is typically caused by expensive bandwidth cost. Furthermore, the volatility and uneven coverage of mobile networks may force the mobile nodes to frequently join and leave such networks, commonly known as *node churn*, which may significantly and drastically affect the performance of the deployed P2P overlay. Indeed, it has been shown that many P2P overlays cannot cope and simply collapse under high churn rate as the proportion of mobile nodes increases in the network [48, 62]. Furthermore, the communication link may be impacted by high packet loss rate and bandwidth fluctuations in mobile networks which make it much more challenging to maintain a stable P2P overlay on the mobile network. Finally, the mobile handset devices also have limited battery power. Therefore, P2P systems, if deployed in a mobile network, are required to minimise their network overhead to prolong device battery life and minimise bandwidth consumption while coping with high levels of node churn.

These inherent properties of mobile networks and their imposed limitations play a major role in influencing the performance of a P2P overlay in mobile networks. It is, therefore,

essential to understand and investigate the effect of these limitations on the performance of different P2P overlays. There are a small number of existing approaches reporting the effect of these limitations in simulated environments. However, all these approaches have limited scopes and are not extensive in nature. Furthermore, NAT (Network Address Translation) is a major barrier to the adoption of P2P overlays in mobile networks. To the best knowledge of the author, no existing research has integrated the NAT functionality within a simulated environment to investigate its impact on the corresponding P2P systems.

This thesis provides a thorough and extensive investigation of the effect of different limitations imposed by mobile networks on existing popular structured P2P overlays in a simulated environment where the limitations have been emulated with different simulation parameters and their selected values. The investigation reveals the potentially best performing P2P overlay(s) that are suitable to be used in mobile networks. In addition, the thesis describes how the NAT functionality has been integrated with the chosen simulation framework. It then studies the effect of NAT on the P2P overlays utilising the same simulation parameters that emulate the behaviour of a mobile network and identifies the potentially best candidate P2P overlays for such setting. Finally, this thesis proposes an adaptive routing scheme to allow any node in the chosen P2P overlays to switch between two routing schemes based on their lifetimes and investigates the suitability of the proposal by simulating experiments utilising the chosen simulation parameters in a NATed churn-intensive network.

## 1.2 THESIS STATEMENT

P2P overlay networks offer major advantages to users of traditional wired networks such as scalable, fault tolerant and self-managing infrastructure which does not exhibit single points of failure. This thesis investigates the feasibility of extending P2P in churn intensive networks such as mobile networks so that users can take advantages of both technologies:P2P and mobile networks.

This thesis experiments with the performance of various structured P2P overlays regarding bandwidth consumption and high churn which are the two most crucial constraints of mobile networks. Furthermore, NATs are a major barrier to the adoption of P2P overlays in mobile networks.

The thesis implements a general NAT traversal approach for ring based overlays without the use of a single server and analyses the performance of selected P2P overlays under NATs. The presence of NATs and also the level of churn in a network influence the routing techniques used in P2P overlays. In addition, this thesis proposes an adaptive routing scheme combining recursive and iterative lookups where nodes can switch between recursive and iterative routing depending on the rate of churn and NAT connectivity.

## 1.3 RESEARCH OBJECTIVES

The key objective of the thesis is to identify the best P2P overlay(s) for using in NATed churn intensive networks such as mobile networks and to strengthen and improve the performance of the P2P overlay(s) using an adaptive routing mechanism. To achieve these goals, the following research objectives have been determined.

- Investigate a number of structured P2P overlays, considering the requirements that are essential to enable P2P in NATed churn intensive networks, and identify the most important parameters of the overlays that affect their performance.
- Provide a comparative performance analysis of a number of structured P2P overlays by utilising the most suitable configuration and evaluate the performance under high levels of churn according to selected performance metrics such as lookup success rate, hop count and bandwidth consumption. Based on the result of the analysis, the P2P overlay(s) will be selected and then will be used for the subsequent research.
- Implement a Network Address Translation (NAT) mechanism to simulate NATed environments in the chosen P2P simulation framework.
- Analyse and compare the performance of the chosen P2P overlay(s) under churn intensive NATed and non-NATed environments.
- Evaluate the performance impact of different routing protocols of P2P overlays in presence of NAT.
- Propose an optimised routing protocol to achieve an improved performance in a structured P2P overlay for NATed churn intensive networks.

## 1.4 CONTRIBUTIONS

The main contributions of the thesis are as follows.

- **Identification of the most suitable configuration of a number of structured P2P overlays:** A P2P overlay has a number of configuration parameters that affect the performance of the overlay for different scenarios. The parameters of Chord, Pastry, Kademlia, Broose and EpiChord DHTs have been evaluated in high churn environments and the best performing configuration has been selected on the basis of maximum lookup success ratio and minimum bandwidth consumption to be used in churn intensive environments such as mobile networks.
- **Comparative performance analysis of a number of structured P2P overlays under churn:** After identifying the best parameters of the chosen P2P overlays, the most

suitable overlays have been finalised for using in mobile networks. For this, a comparative performance analysis has been carried out considering practically relevant scenarios such as high churn and bandwidth consumption.

- **Implementation of Network Address Translation (NAT) and a NAT traversal mechanism in OverSim:** NAT functionalities have been implemented in the chosen P2P simulation framework to integrate private networks with public networks and simulate P2P overlays over the combined networks. To traverse NATs, a NAT traversal approach, UDP hole punching without a rendezvous server, has also been implemented.

- **Performance evaluation of a number of structured P2P overlays under NATed and non-NATed networks:** The performance impact of Chord, Kademlia and EpiChord P2P overlays under churn intensive NATed and non-NATed environments has been evaluated and compared to determine the best suitable overlay in such heterogeneous environments.

- **Performance impact of iterative and recursive routing methods under churn in NATed environments:** The presence of NATs in a network influence the routing techniques used in P2P overlays. Recursive routing is more resilient to IP connectivity restrictions posed by NATs but not very robust in high churn environments. Whereas, iterative routing is more suitable to high churn networks, but difficult to use in NATed environments. Therefore, the performance impact of Iterative and Recursive routing has been analysed under churn and NATed environments.

- **Novel churn aware routing protocol:** In order to achieve an improved performance in a structured P2P overlay for NATed churn intensive networks, an adaptive churn aware routing protocol has been implemented. The main idea of the protocol is to switch between iterative and recursive routing techniques according to the current level of churn of a node.

## 1.5 THESIS STRUCTURE

The rest of the thesis is structured as follows.

After this introduction, Chapter 2 *"Background and Related Work"* gives a brief background on P2P overlays especially on DHT based structured P2P overlays. This chapter also provides a list of challenges to adopt P2P in mobile and wireless networks. Among the challenges mentioned, this thesis focuses on three main issues: high churn rate, data consumption and NAT restriction. It also presents a brief overview of NAT that is a common barrier for any private network. To make a proper P2P connection in such a network, a suitable NAT traversal technique is necessary. This chapter also discusses related work on existing NAT traversal

techniques and P2P solutions regarding mobile networks. A review of various approaches to implement P2P overlays in wireless cellular networks are presented and discussed.

Chapter 3 *"Performance Evaluation of Structured* P2P *Overlays under Churn"* presents a performance evaluation of Chord, Pastry, Kademlia, Broose and EpiChord in the presence of high churn and investigates their suitability for mobile networks. Firstly, the parameters of each overlay have been evaluated in high churn environments and the best performing configuration has been selected on the basis of maximum lookup success ratio and minimum bandwidth consumption. After tuning the best parameters, each overlay has been compared with each other to identify the most suitable overlay to be used in churn intensive networks.

Chapter 4 *"Performance Evaluation of Structured* P2P *Overlays under NAT"* evaluates the performance of Chord, Kademlia and EpiChord under NATed and non-NATed networks. It further compares the performance of iterative and recursive routing algorithms of Chord and Kademlia over NATs. Implemented NAT functionalities and the NAT traversal mechanism, the UDP hole punching technique also have been discussed.

Chapter 5 *"Churn Aware Routing Protocol in NATed Networks"* presents an adaptive churn aware routing protocol that improves the performance of a P2P overlay over a fixed (iterative or recursive) routing method. It also analyses the performance of the proposed protocol.

Chapter 6 *"Conclusions"* first summarises the main contributions and the findings of this thesis. The limitations of the thesis are also highlighted. Finally, the thesis is concluded by presenting a number of directions for future work.

# 2

# BACKGROUND AND RELATED WORK

*This chapter provides some background and discusses a number of existing works relevant to this thesis. It also emphasises the main challenges to the deployment of* P2P *in mobile networks. In addition, Network Address Translation (NAT) imposes a great barrier with regard to mobile networks. Hence, different NAT Traversal techniques have been discussed. Finally, an overview of a number of prominent* P2P *systems proposed for mobile networks is presented.*

## 2.1 INTRODUCTION

The main goal of this chapter is to provide relevant background information to assist in the understanding of the following chapters. Section 2.2 describes the background of P2P overlay networks, mainly the *Structured* P2P. Next, an abstract overview of a number of established DHT based Structured P2P overlays, namely Chord [74], Pastry [67], Kademlia [53], Broose [26] and EpiChord [47] are presented in Section 2.3. The information presented in this section forms the base required to understand the analysis performed throughout the thesis.

Section 2.4 discusses the main challenges of deploying P2P in churn intensive NATed environments such as mobile networks. It also presents related work on churn issues and data consumption. NAT is another major obstacle in P2P as it breaks the original model of end-to-end connectivity across the Internet. Section 2.5 presents the basic NAT terminology and NAT Traversal techniques. The routing method has an important effect on NAT; therefore, Section 2.5.2 discusses the impact of different routing schemes over NAT. There are a small number of existing works on structured P2P overlays with NAT which are described in Section 2.5.3. Section 2.6 discusses a number of approaches to implement P2P overlays in mobile networks.

A detailed overview of different P2P simulators is presented in Section 2.7. Among the presented simulators, OverSim [3] has been selected as the preferred simulation framework to carry out experiments presented in this thesis. The chapter then concludes with a brief summary in Section 2.8.

Throughout the thesis, the terms 'node' and 'peer' have been used interchangeably. Similarly, the term NATed node or NATed network have been used for a node or a network which is under a NAT.

A *Peer-to-Peer (P2P)* network is defined as a multi-node communication network that has neither central control nor coordination. Each participant (either a computer or a mobile device) in a P2P network is simply referred to as a *peer* and the data available on the network is distributed among all the peers on the network. Each peer simultaneously acts as a client and a server and has equivalent capabilities and responsibilities. Additionally, each peer can join and leave the system at any arbitrary time.

A typical P2P architecture creates an overlay on top of the existing IP Network which is the base for data delivery. Figure 2.1 shows an overlay network situated on top of a physical network. The overlay orientation is a logical view; therefore, it is not necessary to directly form a symmetrical view of the physical network. With the rapid growth of the Internet and the advancement of computing technology, peers provide much more information and computing resources than from a limited number of centralised servers. Overlay networks provide benefits to better utilise the Internet information and resources, as routing in overlay networks is very flexible and on the basis of different parameters, they can detect and avoid network congestion quickly. Due to the flexibility in routing, the peers in overlay networks are highly connected to each other. Peers can communicate with each other using the overlay while the physical network connections exist.

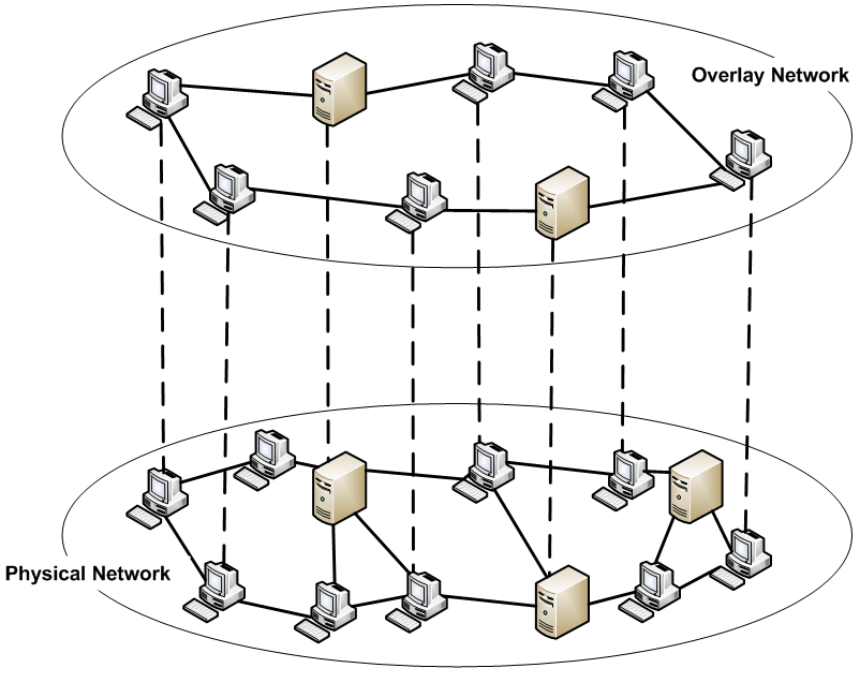



Figure 2.1: Overlay network.

P2P architectures can be broadly categorised as *Unstructured* and *Structured* based on the overlay topology formation.

### 2.2.1 *Unstructured P2P Overlay*

As this type of overlay is not the focus of this thesis, the category is only included briefly for completeness. In an *Unstructured P2P overlay*, there is no particular structure and the links between peers are established arbitrarily. For a lookup query, a peer in this overlay can use *Flooding* or *Random Walk*. Flooding makes each peer, upon receiving a query, forward that query to all its neighbours. Examples of P2P overlays deploying this technique are Gnutella [65] and FastTrack [49]. In random walking, a walker randomly visits different peers and looks for the respective object. Examples of using this approach are Gia [14] and LMS [55].

Such an overlay is easy to construct as the topology of the network is formed randomly. However, there is no correlation between the peers and the resources stored at them, which eventually creates the problem of non-deterministic results; i.e. a peer initiating a query may or may not receive responses from other peers in the overlay. Unstructured P2P overlay also suffers from scalability issues as each peer is required to handle a potentially large number of queries.

### 2.2.2 *Structured P2P Overlay*

To overcome the search inefficiency and scalability issues of an unstructured P2P overlay, *Structured P2P overlays* have been proposed that connect peers using a particular data structure or protocol to ensure that queries are more efficient and data discovery can be guaranteed. Structured P2P overlays utilise a Distributed Hash Table (DHT) to distribute index information about the shared data items among the participating peers. Each node is assigned a unique ID (e.g. a hashed IP address) and is responsible for storing a range of keys which are closest to its node ID. Each data item is also assigned a file ID (e.g. the hash of a file name) and each node is responsible for a particular range of the file ID space. If a node queries a key, the network returns the node ID where the associated file with that key is stored. Some well-known examples of structured P2P overlays are Chord [74], Pastry [67], CAN [63] and Kademlia [53].

The rigid structure makes this kind of overlay more complex, however, they offer a number of key advantages:

- Scalable theoretical guarantees on numbers of hops to find a key.
- Query lookups are deterministic, i.e. if data exists in the network, it will be found.
- Appropriate for large-scale implementations due to high scalability.

Structured P2P overlays can be categorised into *Multi-hop* and *One-hop* overlays according to the maximum number of hops taken by a lookup query.

Multi-hop overlays require multiple hops to pass a request message from the source node to the destination node. According to the node degree with size of overlay, this type of overlay can be further divided into *Logarithmic Degree* and *Constant Degree*. If the number of hops is bounded by $O(logN)$, where $N$ is the number of nodes, it is known as a logarithmic degree overlay. Chord, Kademlia and Pastry are examples of this. In a constant degree overlay, each node has a fixed maximum number of hops, independent of the overlay size. Broose is an example of this type of overlay.

One-Hop or Single-Hop or O(1) overlays are an extreme example of constant overlay in which only one hop is required to deliver a message from the source to the destination. It signifies that each node has an almost complete routing table and thus allows passing messages in one hop. EpiChord is an example of One-hop overlay.

Multi-hop overlays keep their routing table size smaller than the one-hop overlays. In one-hop overlays, each node needs to be aware of every other nodes. This results in a large routing table and increased maintenance traffic. However, the overlay improves the message latency. When the peer churn (i.e. where peers join and leave very frequently in the network) is low and the network size is reasonably small, the one-hop overlays become practical. Whereas, in cases where the network size consists of millions of nodes, or the bandwidth requirement is high, multi-hop overlays might be useful.

## 2.3 Distributed hash table (DHT) based structured P2P overlays

This thesis focuses on structured P2P overlays and experiments with a number of popular DHT based structured P2P overlays such as Chord, Pastry, Kademlia, Broose and EpiChord. A brief discussion of each overlay is presented next.

### 2.3.1 *Chord*

Chord [74] is one of the very first DHT based structured P2P overlays. It is a distributed lookup protocol that addresses the issue of locating nodes that store particular data items. Moreover, Chord is scalable and adapts efficiently in the face of churn.

**Structure:** Chord uses the consistent hash function [40] to generate an $m$-bit identifier for each node and the shared data item on the overlay, where $m$ is a pre-defined system parameter. A node identifier is produced by hashing the node's IP address, whereas a data or key identifier is achieved by hashing the data item. Nodes are ordered to form a ring circle of modulo $2^m$ and the address space ranges from $0$ to $2^m - 1$. A key $k$ is assigned to the first node whose identifier is equal to or follows $k$ (clockwise) in the ring. This node is called the successor node of key $k$ and denoted by *successor (k)*. Figure 2.2 shows a Chord identifier

circle that has three nodes: 1, 4, and 7. The successor of key identifier 1 is node 1, so key 1 would be located at node 1. Similarly, key 2 would be located at node 4, and key 6 at node 7.

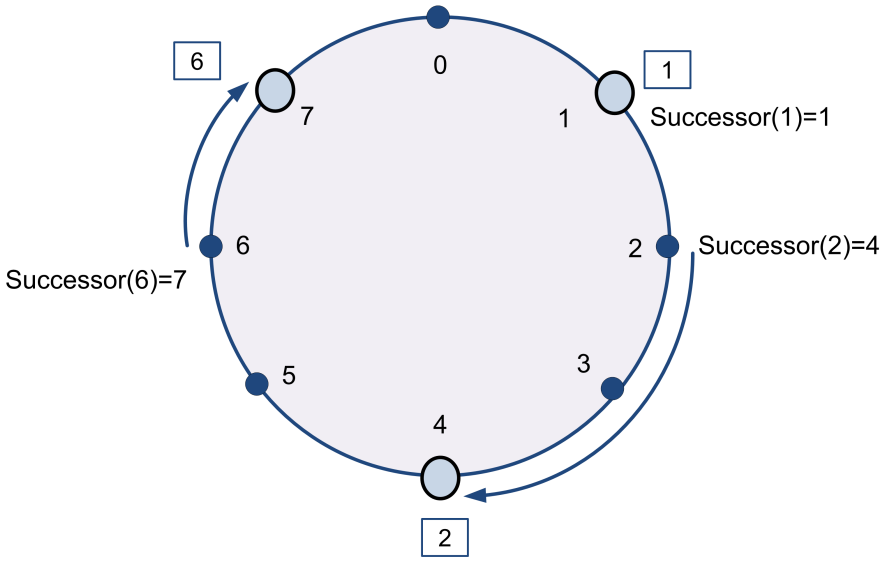



Figure 2.2: A Chord identifier circle with three nodes (1, 4, 6) consisting of three keys (1, 2, 6) where each successor of a key contains the key.

**Lookup:** To route a lookup request, each node maintains a routing table which consists of two parts: the first part is a finger table with (at most) $m$ entries, and the predecessor of this node. The first finger in the finger table always contains the immediate successor of the current node. The second part is a neighbour table that contains the successor list of size $r$. When a lookup is initiated, it first checks its finger table for its successor node of the given key and routes the message to that node. If the node does not know the successor for the given key, it routes the message to a node that is numerically closer than itself. This relaying process continues iteratively or recursively thereafter until the closest node or destination is found.

**Node Join:** When a new node $n$ joins the Chord overlay, it first uses the consistent hashing to obtain its $m$-bit identifier and then contacts a node which is already participating in the Chord overlay, known as a bootstrapping node, to lookup the successor node of its identifier. This successor node $s$ then becomes the new node $n$'s successor. As a result, $s$ changes it's predecessor pointer to the new node $n$. To ensure a correct routing table, the new node $n$ uses the *stabilisation protocol*. Stabilisation protocol is used to keep nodes' successor pointers up to date. It is called periodically in the background at each node. The protocol has two main functions: Stabilise() and Fix_fingers(). The Stabilise() function allows nodes to learn about new nodes and to update their successors and predecessor, whereas the Fix_fingers() function ensures the finger table is current.

**Performance:** To lookup a key, Chord nodes require $O(\log N)$ messages, where $N$ is the number of nodes.

### 2.3.2 *Pastry*

Pastry [67, 11] is another multi-hop structured P2P overlay, with a lot of similarities with Chord. It provides a self-organising routing and location service. Several P2P applications built on top of Pastry are used in global data storage, content distribution and group communication. Two most popular examples are SCRIBE [12, 69] and PAST [20, 68].

**Structure:** Similar to Chord, Pastry also consists of a circular identifier space where each node and data item are assigned a 128-bit identifier with base $2^b$ ($b$ is a configuration parameter of the algorithm), using a consistent hash function. Each Pastry node uses a routing table consists of $\log_{2^b} N$ rows and $2^b - 1$ columns, where the entries in row $n$ contain routing information of all nodes whose identifier has the same $n$ first digits with the present node. In addition to the routing table, each node in Pastry maintains a *Leaf set* and a *Neighbourhood set*. The leaf set L contains the $L/2$ nodes with numerically closest larger nodeIds, and $L/2$ nodes with numerically closest smaller nodeIds, relative to the current node's nodeId. The leaf set is used to store object replica and ensures reliable message delivery. The neighbourhood set M contains the closest nodes in terms of proximity to the current node.

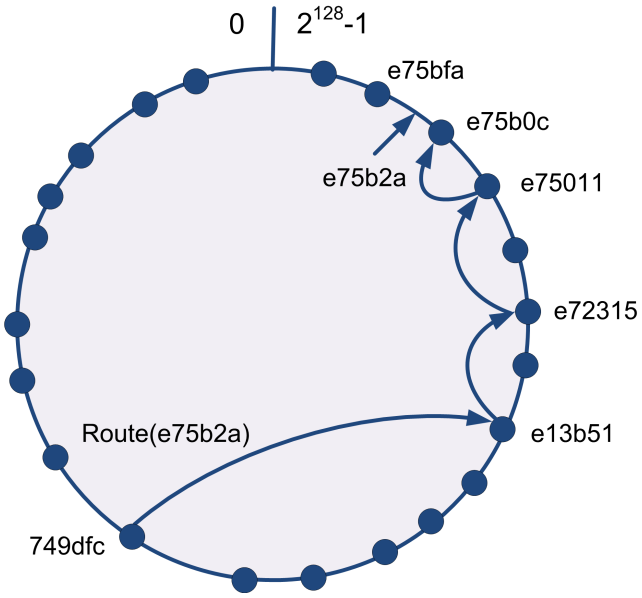



Figure 2.3: Example of a lookup in Pastry. Routing a message from node 749dfc with key e75b2a.

**Lookup:** To find a node with a specific key, the initiator Pastry node first checks its leaf set if it has a node whose nodeId is closest to the key. If a node is found, then it forwards the query directly to the node. If not, the node checks the routing table and the query is forwarded to a node whose nodeId shares a common prefix with the key by at least one more digit. In this way, using $\log_{2^b} N$ steps, where N is the total number of Pastry nodes, the lookup query can reach its destination node. Figure 2.3 shows the path of an example of a query lookup in Pastry. For example, the node 749dfc looks up for a key e75b2a and from its routing table it

gets e13b51, which shares the first digit common prefix with the key. Then e13b51 checks its routing table and finds e72315 that shares first two digits common prefix with the key. This process keep continues until the key is covered by the node e75b0c.

**Node Join:** When a new node joins in the Pastry overlay, it contacts a well known bootstrap node in the network. The bootstrap node returns a list of close nodes to the new node. The new node then issues a join message with its nodeId as the key. One of the existing nodes, that is numerically closest to new node's nodeId, receives the join message and sends its routing table to the new node. All the nodes encountered on the routing path also send their routing tables to the new node. The new node then initialises its routing table based on the received information and also informs any nodes that need to be aware of its arrival.

**Performance:** Pastry can route to any node in $O(logN)$ steps in a network of $N$ nodes.

### 2.3.3 *Kademlia*

Kademlia [53] is the most widely deployed structured P2P overlay [13]. Numerous commercial and open source P2P applications have been implemented using Kademlia, such as Bittorrent [6] and eMule [22]. Kademlia uses a bitwise exclusive or XOR based distance metric to calculate the distance between two nodes rather than the numerical closeness of the two nodes. By using this XOR metric, Kademlia successively finds nodes approximately half as far from the target node.

**Structure:** Kademlia assigns a NodeId to each node in a 160-bit key space and maintains a routing table consisting of $k$-buckets. Each $k$-bucket contains a maximum of $k$ entries with <IP address, UDP port, NodeId> triples of other nodes. The routing table forms a binary tree where nodes are represented as leaves and the $k$-bucket is represented as a series of sub-trees that do not contain the node itself. Figure 2.4 shows a Kademlia routing tree for a node with a nodeId prefix of 001.

**Lookup:** Kademlia utilises two important methods in a lookup process. It sends lookup queries in parallel and exchanges the routing table entries during the same lookups; instead of sending separate requests for maintenance. During a lookup, a peer computes the XOR distance to the destination peer and checks in the corresponding $k$-bucket in its routing table for the closest $\alpha$ nodes. It then sends parallel requests to these $\alpha$ nodes. In return, each node responds with the closer nodes to the requested identifier by looking at the entries in its $k$-buckets. From the responses, the initiator node again selects $\alpha$ closest nodes and iteratively sends parallel requests to these nodes. This process is repeated until the requested node identifier is found. To improve lookup latency, Kademlia also supports redundancy and caching.

**Node Join:** When a new node joins a Kademlia overlay, it has to know an existing bootstrap node, and performs the first lookup to it. The bootstrap node then inserts the new node in the

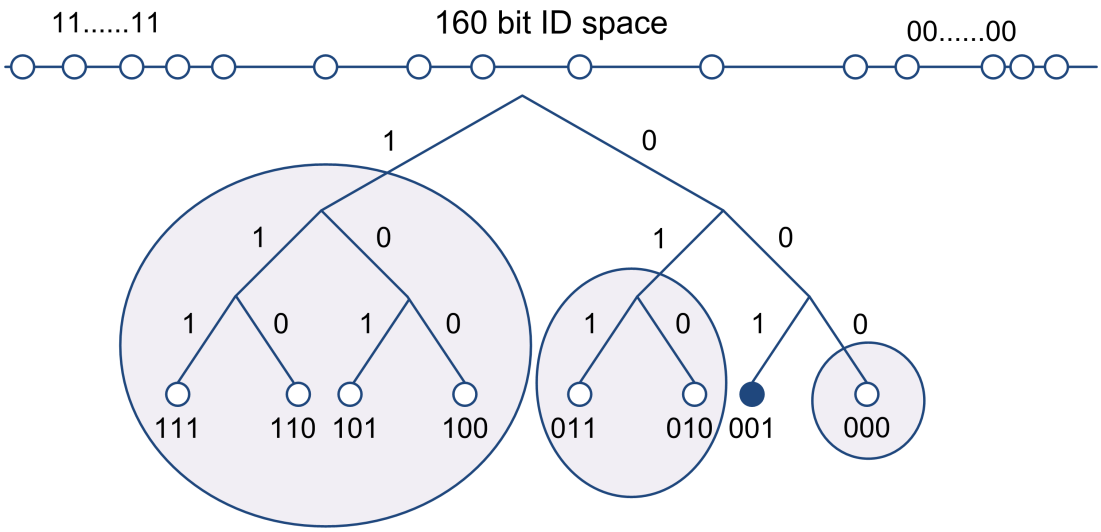



Figure 2.4: Routing table in Kademlia for a node with a nodeId prefix of 001. The ovals show the sub-trees in which node 001 must have a contact.

appropriate k-bucket according to the new node's nodeId and replies with information about other nodes to the new node. Once the new node receives information about other nodes, it performs lookups to random nodes on each k-bucket, and exchanges routing table entries.

**Performance:** Kademlia's routing performance is consistent and achieves a lookup latency of $O(\log N)$, where $N$ is the number of nodes.

### 2.3.4 *Broose*

Broose [26] is a P2P protocol inspired by Kademlia and is based on the De-Bruijn topology that allows a DHT to be maintained in a loose manner. It combines the advantages offered by the De-Bruijn topology and Kademlia in different aspects of topology maintenance and lookup performance.

**Structure:** Similar to other DHT approaches, a key in Broose is an $n$-bit (where $n$ can be 128 or 160) binary string. An ID in this 160-bit key space is randomly chosen by each node. However, as in Kademlia, the key space is not split among different nodes, instead, an associated information is determined using a specified metric and stored on the nodes which are closer to the key based on that metric. Each node maintains buckets of possible nodes which are in charge of specific parts of the key space. In Kademlia, each node requires to maintain $O(\log N)$ buckets whereas each node in Broose only needs to maintain three buckets.

Broose utilises the De-Bruijn graph which, for $N = 2^n$ nodes, can be defined as follows. Each node $u$ is identified with an identifier $u[1, n]$ which essentially represents an $n$-bit binary string. Two successors of the node are identified by shifting one bit of the identifier to the left ($u << 1$) and then appending a $0$ and $1$ on the right-most bits. Hence, two successors are identified as $s = u[2, n]0$ and $s' = u[2, n]1$. Similarly, the predecessor can be identified by

shifting one bit to the right ($u >> 1$) and then appending a 0 or 1 on the left-most position. That is, the two predecessors can be identified using: $p = 0u[1, n-1]$ and $p' = 1u[1, n01]$. This graph structure ensures that each node has a constant in-degree and out-degree of 2 and a lookup can be performed by shifting bits to right from one predecessor to another or shifting bits to left from one successor to another until the node with the desired key is found. Broose offers a generalised approach for lookup by shifting bits to left or right utilising a simple routing table consisting of three buckets. Like Kademlia, no topology maintenance is needed with regard to join and removal of a node.

**Lookup:** A lookup process in Broose utilises a right-shift by default whereas a reverse lookup is carried out by the left-shift procedure. To lookup for a key $w$ initiated by a node $u$, a bucket $K$ is initialised with the node itself $\{u\}$. Then, the node $u$ estimates the distance $d$ which depicts the number of hops to a node storing $w$ from $u$. The distance $d$ is calculated using the following formula:

$$d = l + 1$$

Here, $l$ represents the length of the longest common prefix of the identifiers of nodes in the bucket of the querying node. An iteration counter $d_K$ is initialised with $d$ (i.e. $d_K = d$) and the following steps are repeated until $d_K$ reaches to 0:

- $u$ contacts one to $\alpha$ (where $\alpha$ is a protocol parameter) nodes in $K$ for a right-shifting lookup on $w$ for $d_K$ hops. Each queried node replies with its bucket containing predecessors on $w$ for $d_K$ hops;
- if the reply is for $d_K$ hops, $K$ is replaced by the nodes contained in the bucket and $d_K$ is decremented by one;
- if the reply is for $d_{K+1}$ hops, the nodes contained in the bucket is appended to $K$;
- if the reply is for more than $d_{K+1}$ hops, the reply is ignored.

**Performance:** Broose utilises a fixed size of $O(k)$ routing table where $k$ is a protocol parameter, and represents an association of $k$ number of nodes. The lookup performance of Broose is similar to Kademlia and can be performed by contacting only $O(logN)$ nodes.

### 2.3.5 *EpiChord*

EpiChord [47] is a DHT algorithm where peers maintain a full routing table and ideally approach $O(1)$ hop lookup performance, compared to the $O(logN)$ hop performance offered in many multi-hop networks.

**Structure:** EpiChord is based on the Chord DHT and organised as a one-dimensional circular space, with each node assigned a unique node identifier. The node responsible for a

key is the node whose identifier most closely follows the key. In addition to maintaining a list of the $k$ succeeding nodes, EpiChord also maintains a list of the $k$ preceding nodes and a cache of nodes. Nodes update their cache by observing lookup traffic. Therefore, nodes add an entry any time they learn of a node not already in the cache and remove entries which are considered dead. In general terms EpiChord can be thought of as Chord with a cache of extra node addresses. This results in lookup performance approach one hop with a well-populated cache. The performance may drop to that of Chord, $O(\log N)$ hops, under high churn.

EpiChord uses parallel requests [45] to avoid sending redundant queries. It sends $p$-way parallel requests directed to nodes nearest to the destination node that should be responsible for the desired data. It allows the querying node to receive all information related to the query path, and hence, update its cache with new entries. Such an EpiChord network with at most $p$ parallel queries per lookup is called a $p-way$ EpiChord network.

**Lookup:** EpiChord utilises an iterative lookup algorithm. To lookup a resource with the key *id*, a node $x$ will initiate $p$ queries in parallel - one query to the node immediately succeeding *id* and the $p-1$ queries to the nodes preceding *id*, as shown in Figure 2.5. When queried, a node will respond as follows:

- If it owns *id*, it will respond with the value associated with *id*, and information about its immediate predecessor and successor.
- If it is a predecessor of *id* relative to the querying node, it will provide information about its immediate successor and the $l$ best next hops towards the destination.
- If it is a successor of *id* relative to the querying node, it will provide information about its immediate predecessor and the $l$ best next hops towards the destination.

Here $l$ and $p$ are both system parameters. When a reply is received, further queries are dispatched in parallel if the querying node learns about any node closer to the target id than the best successor and predecessor nodes that have responded already.

**Node Join:** A node that wants to join the EpiChord network will know at least one bootstrap node that already exists in the network. The new node performs the join operation using two steps:

- It makes a regular iterative lookup for the successor to its NodeId in the ring with the help of the bootstrapping node.
- When the successor is found, the new node sends a request to its successor to initialise its cache. It also generates a join token with its successor and passes the token to its predecessor.

Once a node finds its successor and predecessor, and obtains a full cache transfer, the join is complete.

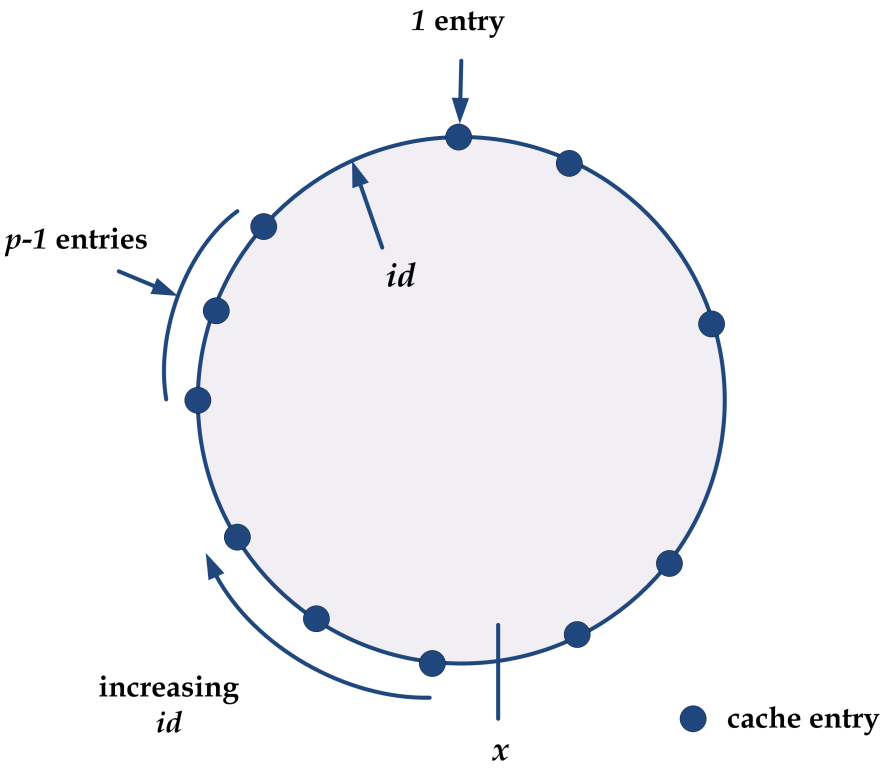



Figure 2.5: Basic lookup mechanism for node x for a lookup of *id* in EpiChord.

**Performance:** The performance of EpiChord has been measured in two workloads: *Lookup-intensive* and *Churn-Intensive*. A workload is considered as Lookup intensive if the *Lookup rate >> Rate of node join*, and it is churn intensive if the *Lookup rate << Rate of node join* [47]. EpiChord achieves $O(1)$-hop lookup performance under lookup intensive workloads, and at most $O(\log n)$-hop lookup performance under churn-intensive workloads.

## 2.4 Challenges of deploying P2P in mobile networks

Current P2P research focuses mostly on fixed networks. Adopting P2P for NATed churn aware environments containing mobile devices with an array of wireless technologies imposes a significant number of challenges. A non-exhaustive list of challenges is:

- **High Churn Rates** - Churn refers to the rate of peer joins and leaves in a mobile network. High churn rate can be caused by variation in signal strength, data calls being interrupted by voice calls, and/or simply discharged batteries.
- **Energy/Data Consumption** - Battery power on mobile devices is limited. Moreover, sending/receiving data on cellular networks is energy intensive and hence, the number-/size of P2P maintenance messages should be as small as possible.

- **Network Address Translation (NAT) and Firewall** - NAT routers and Firewalls are deployed by mobile network operators enabling phone initiated outgoing connections only. This poses difficulties for P2P networks to connect one node to another.
- **Bandwidth Usage** - Another important fact is that mobile devices have their bandwidth limited by mobile networks. Even though, wireless technologies have improved a lot; the developers should take this into account while developing mobile P2P applications.

There are some other challenges such as *Network Access Methods*, *Nodes and Resource Discovery* and *Security*, however, this research focuses mainly on the churn, data consumption and NAT traversal issues. The research works related to churn, bandwidth consumption and NAT are reviewed next.

### 2.4.1 *Churn Issues*

DHT based structured P2P offers high scalability and improved data lookup efficiency, however, the performance can be affected significantly by frequent join and leaves of mobile nodes, i.e. churn. Churn imposes two main problems for P2P routing. Firstly, it makes the routing table out-of-date and the stale entries in the table point to neighbour nodes that are either dead or have already left the network. This causes multiple round trip time to find a lost lookup message and hence, the system experiences expensive lookup time-outs. Secondly, as new mobile nodes join the P2P overlay and stale entries become obsolete, the routing table needs to be refreshed which does not happen very quickly. As a result, the system experiences inefficient routing, increasing lookup latency and high overlay maintenance overheads. If the proportion of mobile nodes increases in the network, many structured overlays simply collapse under high churn rate [48], [62]. Therefore, the impact of churn needs to be considered when designing a P2P system targeted for mobile churn intensive environments. A number of approaches are available that attempt to deal with churn issues.

Stutzbach and Rejaie [75] showed that the node life time or churn distribution could accurately be modelled by a Weibull distribution but not by the Poisson or Pareto distributions. They also examined churn at two levels: *Group-level characterisation* and *Peer-level characterisation*. In the group-level characterisation, the behaviour of all participating peers are captured collectively, whereas in the peer-level characterisation, the behaviour of specific peers are captured across multiple appearances in the system over time. To characterise these aspects of peer dynamics, they investigated three different P2P systems: Gnutella, Kad and BitTorrent. The experiment results revealed that the group-level characterisation of churn exhibited similar behaviour among all three applications, but peer characterisation in BitTorrent were significantly different.

Rhea *et al.* [64] stated that existing DHT implementations could not handle high levels of churn. They identified three factors that improved the DHT performance under churn which

were: *Reactive versus Periodic Recovery from failure*, *Lookup Timeouts* and *Proximity Neighbour Selection (PNS)*. The authors used periodic recovery in their design, as reactive recovery might lead to a feedback cycle that crowded the network. For calculation of lookup timeouts, they showed that TCP-style timeout calculation achieved the best results when compared to alternative *Fixed Timeout* or *Virtual Coordinate* techniques. For PNS, they considered Global Sampling, *Neighbours' Neighbours (NN)* and *Neighbours' Inverse Neighbours (NIN)* techniques; and found that combining Global Sampling with any other technique was more effective than simply selecting NN or NIN.

Wu *et al.* [76] provided directions to design a DHT under churn. They conducted an analytical study of three features - lookup strategy, lookup parallelism and lookup key replication - to improve the DHT lookup performance under churn. They compared the performance of two lookup strategies: *Recursive Routing (RR)* and *Iterative Routing (IR)*. Furthermore, they proposed two enhanced alternatives. One is a two-phase routing strategy (RR+IR), where the RR is used in the first phase and if RR fails, IR is conducted as the second phase. Another one is Recursive Routing with ACK strategy (RR+ACK), where an ACK is sent by an intermediate node, containing the address of the next hop to the originator along with the RR. Through simulation, the authors showed that the RR+ACK technique performs better than others in different churn conditions. For lookup parallelism, the authors showed that a parallelism degree of two or three can increase the overall lookup performance. *Lookup Key Replication* is another important feature in which one data key is replicated in multiple peers. They studied two replication policies: replication without repair mechanism and replication with repair mechanism and found that the necessity of a repair mechanism depends on the peer lifetime distribution. In *Exponential Distribution*, a repair mechanism is required no matter what the churn level is; however, in *Pareto Distribution*, if there is a very heavy tail, simple replication is enough to guarantee a sufficient long lifetime of the replicated key.

Ou *et al.* [60] evaluated the performance of a P2P system in mobile environments in the presence of different levels of churn. Kademlia [53] was used as the underlying DHT algorithm and P2P *Protocol (*P2PP*)* as the signalling protocol in their design. After conducting simulation and implementing a prototype on a mobile platform, they concluded that a lookup parallelism degree of three and resource replication degree of three were good enough to make the system robust under various levels of churn and ungraceful leaving of nodes. They also showed that a good success ratio could be achieved by parallel lookup even without using KeepAlive messages. However, they only simulated the overlay with 400 nodes and they did not compare the performance of the overlay with any other overlays.

Li *et al.* [48] presented a *Performance Vs. Cost (PVC)* framework to compare the effects of different features of various DHTs under churn. The authors simulated Chord, Kademlia, Kelips [33], OneHop [32] and Tapestry [79]; and used PVC to determine the most important features to handle churn. The results revealed some important observations about DHT design

choices, such as, increasing the routing table size is better than faster stabilisation; parallel lookup is more efficient than stabilisation; and learning about new nodes from lookup can replace stabilisation.

### 2.4.2 *Data Consumption*

Energy consumption is closely related with how much data is transferred, which ultimately dictates the data consumption of a particular network. Hence, the consumption of energy and data is another challenging issue in mobile devices. As the battery power of a mobile device is limited and the mobile phones may not be always online due to user requirements and user mobility outside of the network coverage, one main concern of the developer should be developing the mobile application in an energy optimised way. Some contributions have focused on the energy consumption perspective.

Gurun *et al.* [34] assessed the energy consumption utilising a structured P2P chat application on a *Personal Digital Assistant (PDA)* device and showed that it was feasible to develop lightweight P2P protocols and applications on limited-power, embedded devices, such as PDAs or mobile phones. They also specified that the power savings could be improved if overlay messages were batched and sent periodically.

Nurminen and Noyranen [58] carried out measurements of energy consumption of BitTorrent [16] on hand-held devices and their results indicate that P2P content sharing on mobile phone is feasible from the energy consumption point-of-view, which is in line with previous studies. They also showed that active uploading and downloading of contents did not consume extra energy while acting as a full peer. However, it is recommended that the uploading should be terminated as soon as the downloading was finished to save energy. For a comparative analysis, they performed the measurements for both 3G and WLAN access networks.

Kelényi and Nurminen [41] installed a DHT client in mobile devices and measured the rate of energy consumption of mobile devices over Wi-Fi. They reported that a mobile device could only last a couple of hours when it was acting as a full peer. The same group of researchers proposed an energy optimised, load balancing protocol for Kademlia based DHT which allows mobile devices to reduce energy consumption by selectively dropping incoming messages over WLAN [43]. Their measurement study revealed that energy consumption could be reduced as much as 55% by dropping 50% of incoming messages. However, dropping 70% of the request could not achieve any significant benefit.

Ou *et al.* [59] implemented a prototype to evaluate the feasibility of mobile nodes acting as fully fledged peers and measured the power consumption. The consumption results showed that the WLAN access mode consumed less power than the UMTS access mode. Furthermore,

in the UMTS access mode, the protocol packets of 200 bytes or less were the most energy efficient.

Kelényi *et al.* [44] used broadband routers to improve the energy-efficiency of BitTorrent downloads to mobile phones. It was shown that using a proxy server could save energy for mobile phones.

Network Address Translation or NAT is another major challenge for deploying P2P. The next section elaborates on NAT and existing NAT traversal techniques offering transparent traversal abilities to keep the P2P connection alive.

## 2.5 NAT AND NAT TRAVERSAL

NATs are used to provide a mapping of a single public IP address onto several end systems on a private network, thereby allowing many computers in the private network to access the Internet using the single public IP address [21]. The basic idea is to let a NAT based router replace the IP header of the packets and maintain a mapping table that contains the address information in outgoing and incoming messages. Figure 2.6 shows the general NAT approach.

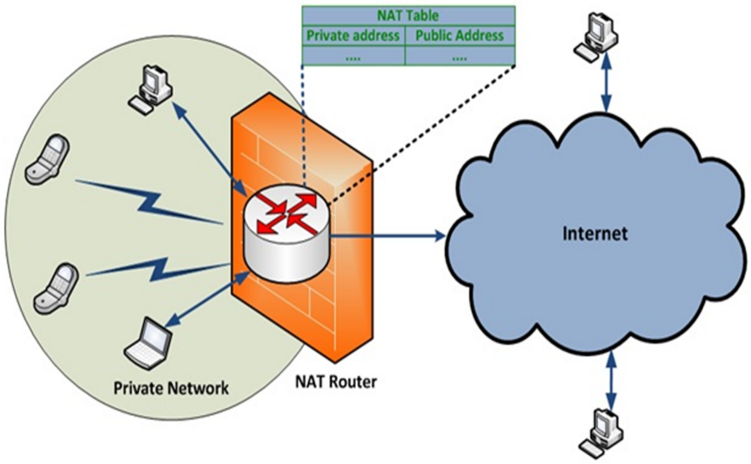



Figure 2.6: Network Address Translation (NAT).

Depending on the address mapping schemes, there are four types of NAT: Full Cone NAT, Restricted Cone NAT, Port Restricted Cone NAT and Symmetric NAT [66].

- **Full Cone NAT:** In a full cone NAT, all requests from an internal IP address and port are mapped in the same public external IP address and port. Additionally, other external hosts can send a packet to the internal host using the mapped external address. Figure 2.7 shows an example of full cone NAT.
- **Restricted Cone NAT:** A restricted cone NAT is like a full cone NAT; the only difference is that an external host with the IP address X can send a packet to an internal host only

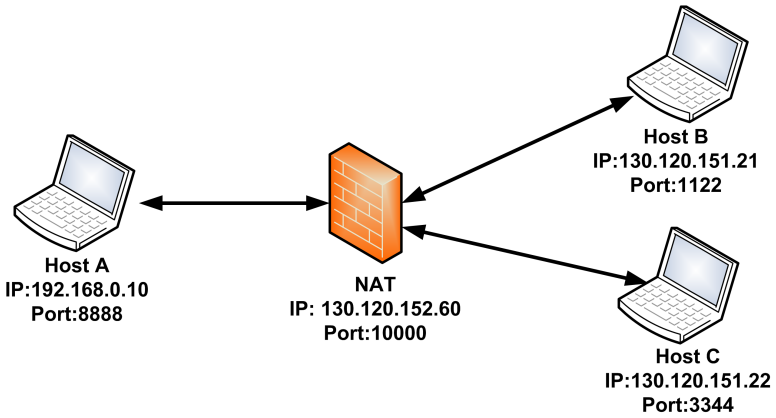



Figure 2.7: Full Cone NAT.

if the internal host previously sent a packet to the host with the IP address X. Figure 2.8 shows the behaviour of the restricted cone NAT.

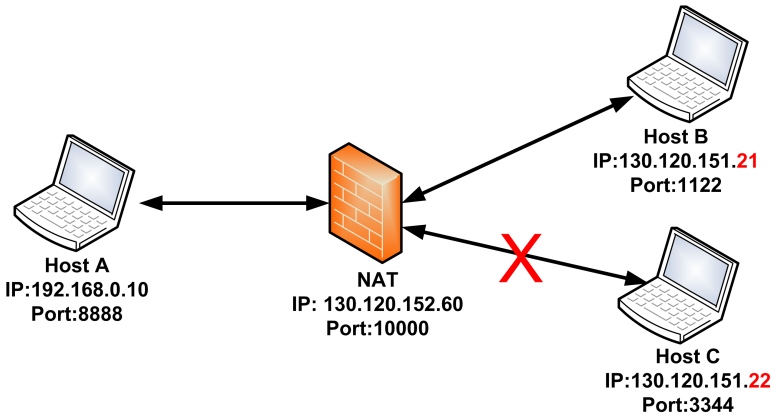



Figure 2.8: Restricted NAT.

- **Port Restricted Cone NAT:** A port restricted cone NAT is similar to the restricted cone NAT. Additionally, it requires the external host to use the same port number that the internal host previously used to contact it. Figure 2.9 shows an example of a port restricted cone NAT.

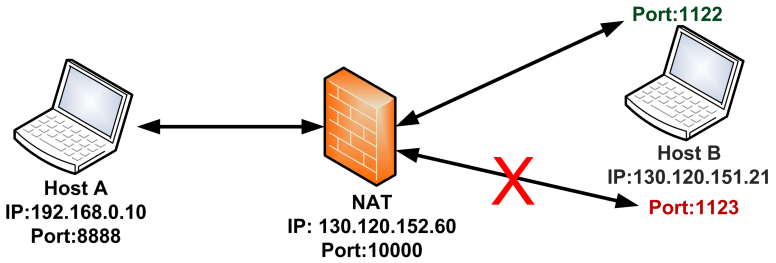



Figure 2.9: Port-Restricted NAT.

- **Symmetric NAT:** A symmetric NAT creates a different IP address and port number mapping according to a session IP and an arbitrarily chosen externally used port number. Figure 2.10 illustrates the behaviour of the symmetric NAT.

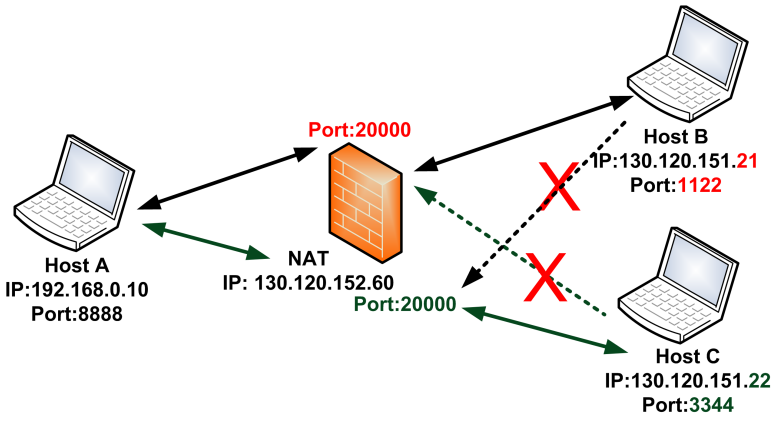



Figure 2.10: Symmetric NAT.

### 2.5.1 *NAT Traversal Techniques*

NAT is suitable for any typical Client-Server communication when the client is on a private network and the server is in the global address realm. A P2P network is different from the Client-Server network. In a P2P network peers have equal positions without any classification of client and server. They are directly connected to other peers and they both act as client and server simultaneously. In NATed environments, any general NAT architecture makes it difficult for two nodes on different private networks to communicate with each other directly. Therefore, it is important for NAT device makers, protocol designers and P2P application vendors to provide smooth and secure two way direct communication, including unsolicited incoming connection attempts for hosts residing in NATed environments.

Different NAT traversal mechanisms have been designed to provide direct communication between peers behind NATs. Some are based on NAT gateway optimised and plugged techniques such as *Universal Plug and Play (UPnP)* [8] and *Application Level Gateway (ALG)* [72], whereas some are based on fall-back (making use of the Client-Server model) approaches, in which they depend on a relay server or a rendezvous server on either or both sides of the NAT gateway. Some examples of such techniques are STUN [66] and TCP/UDP hole punching [25]. Among them, the hole punching is the most robust and practical NAT traversal technique as it works consistently for both UDP and TCP and can be deployed by applications without special requirements or privileges [25].

The next subsections will provide a brief review of a number of currently known techniques for implementing P2P communication over existing NAT devices.

#### 2.5.1.1 *STUN*

*Simple Traversal of UDP through NAT* or *STUN* [66] is a simple client-server protocol that is a well known and the most widely used VoIP NAT traversal solution for UDP traffic. It allows applications to discover the presence and types of NATs between them and the public IP address.

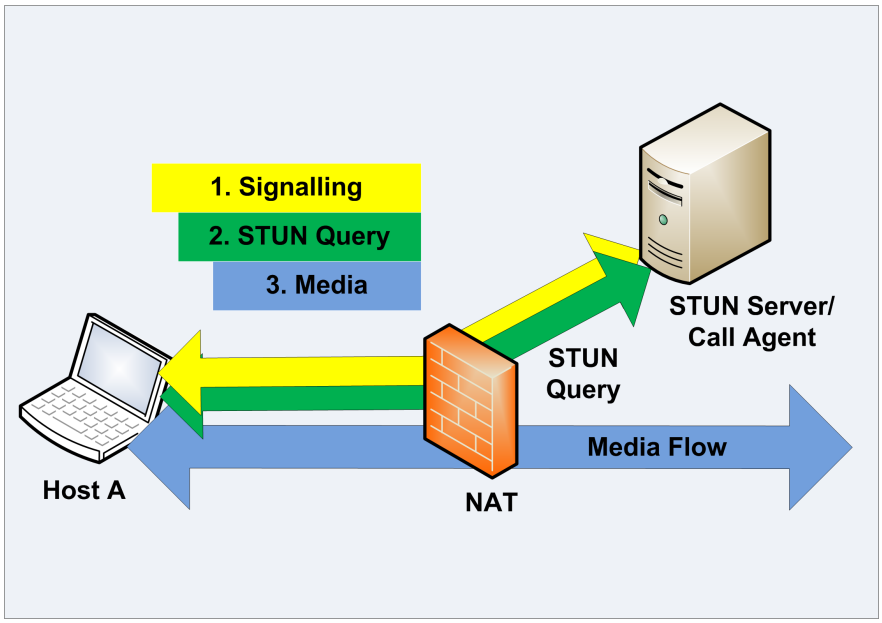



Figure 2.11: STUN.

A possible STUN configuration has been shown in Figure 2.11. The host behind a NAT is called a STUN client, and the server on the external side is called a STUN server. The STUN client sends a Binding Request over UDP to the STUN Server which typically resides in the public address realm. The UDP request packet may traverse several NAT devices to reach the STUN server. The Server discovers the last NAT-modified source address and port and then copies the source address and port into a Binding Response which is then sent back to the STUN client. By comparing the local address and port in the response packet with its own record, the STUN client can discover if it is behind a NAT device.

When the client detects that it is behind a NAT, it does a series of tests to determine its exact type. These tests consist of asking one or two STUN servers to send their responses from different ports and analysing these responses to determine how the NAT has mapped each outgoing request. After detecting the existence and type of the NAT, the client uses the mapping that the NAT allocated for the STUN server to construct its messages.

The main advantage of using STUN is that it does not need any changes to NAT devices. Clients can discover NAT devices automatically. However, it does not support symmetric NATs which are reasonably common. STUN requires client applications to be upgraded to

support STUN and an additional STUN server residing in the public domain. The use of STUN can be hindered due to these reasons.

#### 2.5.1.2 *UDP Hole Punching*

The Hole Punching technique [25] enables direct P2P communications between hosts or peers, even if the peers are both behind NATs, with the help of a well-known rendezvous server. The rendezvous server allows the peers to discover each others' endpoints (IP address and port) so that they can communicate directly.

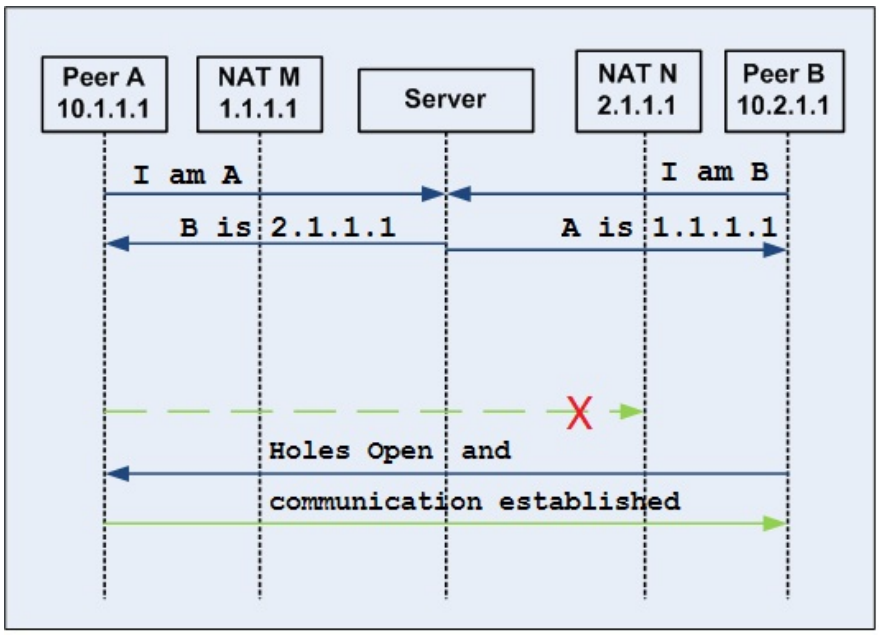



Figure 2.12: UDP hole punching.

Figure 2.12 shows an example of the UDP hole punching process. Suppose peers A and B are both behind different NATs M and N respectively. A and B have established a UDP session to the server and, the NAT M and N will create UDP translation states and assign a temporary external port number for each internal peer. Afterwards, the server will relay the information containing their IP address and port number back to peer A and peer B.

Now peer A sends packets to the public address of the peer B. Since the remote NAT N does not have a translation entry for this connection yet, it drops all messages. Now B sends data packets to the public address of A, to the exact IP address and port where the previous message came from. Although the message of A was dropped, N created a translation entry on its NAT device which now allows the message from B to pass. This enables the UDP connection to be established.

Hole punching is well understood for UDP, but it can also work for TCP communication [25]. To establish a direct P2P TCP connection between peers under NAT is a little more

complex than for UDP. It works in a similar way to UDP hole punching, only with the added complexity of establishing the TCP handshake between the peers.

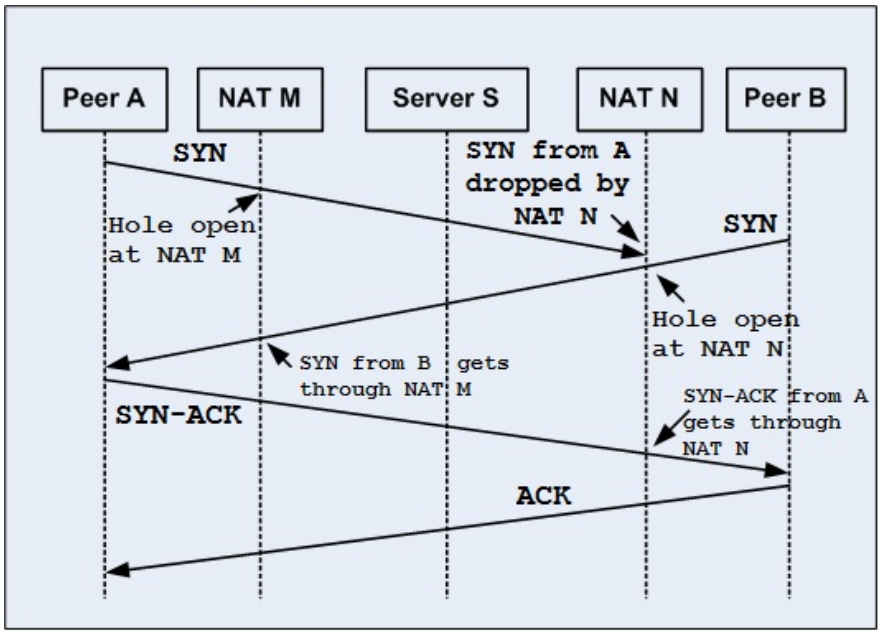



Figure 2.13: TCP hole punching.

Suppose two peers A and B are both behind different NATs as shown in Figure 2.13. Similar to UDP, both peers establish an active TCP connection to the same rendezvous server S. The peers also register their private and public addresses on the server. Each peer's first SYN packet to the other peer creates a 'hole' in its respective NAT. In the situation when A's first SYN packet to B reaches at B's NAT before B's first SYN packet to A reaches at B's own NAT, B's NAT tags A's SYN packet as unsolicited and therefore, drops it. However, B's SYN packet can get through A's NAT successfully because B's public address is recognised by A's NAT as part of the outgoing session to B that A had already initiated.

Hole punching preserves the transparency of NAT and works even with multiple levels of NATs. However, it also has some disadvantages. By introducing a server, it turns the part of a P2P network into a Client-Server model. It adds overhead to the bandwidth and increases communication latency as well.

#### 2.5.1.3 *TURN*

In some scenarios, a direct connection between nodes behind different NATs is impossible, particularly if nodes are behind a symmetric NAT. In these cases, it is necessary to relay the communication via an external intermediate node. *Traversal Using Relays around NAT (TURN)* [50] is a protocol that provides a relaying service via a TURN server. As this approach makes heavy use of relay techniques, it works in almost every imaginable situation. However, this service is expensive to maintain as it is required to maintain a robust and high-capacity TURN

server. Consequently, in most situations, TURN is used as a fall-back mechanism for other methods, such as hole punching, to solve the NAT connectivity problem. This approach works both for TCP or UDP.

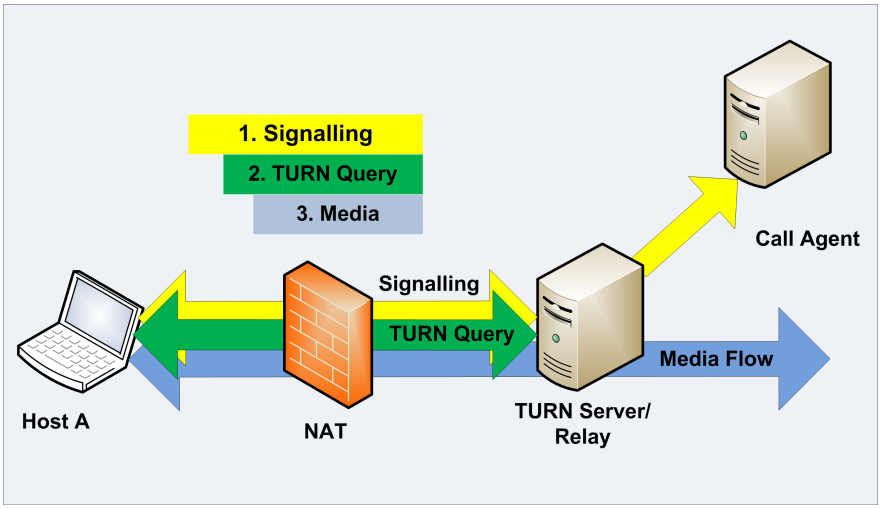



Figure 2.14: TURN.

The principle of TURN is to provide a client with a public Internet address, which are usually hidden by NAT devices. This allows the clients to request a public address and port from the TURN server. The TURN server communicates to the TURN client with some requests/responses besides relaying and the server is transparent to external peers. Even so, one client behind the NAT and the TURN server does not receive the messages from the third (other) peers.

Unlike STUN, TURN does not allow direct connectivity between two hosts behind their NATs. Although TURN can traverse each type of NAT including symmetric NAT, the TURN protocol will burden the public TURN server with heavy loads and might cause delays. Due to its associated high cost, TURN will be the last resort to use in practice. The Figure 2.14 shows the TURN architecture.

#### 2.5.1.4 *STUNT*

*STUNT (Simple Traversal of UDP through NATs and TCP too)* [31] is a protocol that extends STUN to include TCP. It includes two approaches for traversing NATs. The first approach is known as STUNT#1, which has been illustrated in Figure 2.15. The figure shows that the two peers A and B are behind two different NATs M and N respectively and the STUNT server is on the public Internet. In this approach, both peers send an initial SYN with a high enough TTL to cross their own NATs, however, the TTL is low enough that the packets are dropped in the network (once the TTL expires). The peers learn the initial TCP sequence number by listening for the outbound SYN over PCAP or a RAW socket. Both peers inform the STUNT server of their respective sequence numbers and the STUNT server replies with a SYN-ACK

(spoofed) to both peer with the appropriate sequence numbers. The ACK completes the TCP handshake and therefore goes through the network as usual.

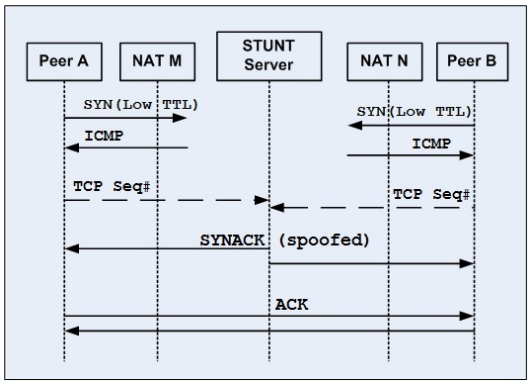



Figure 2.15: STUNT#1.

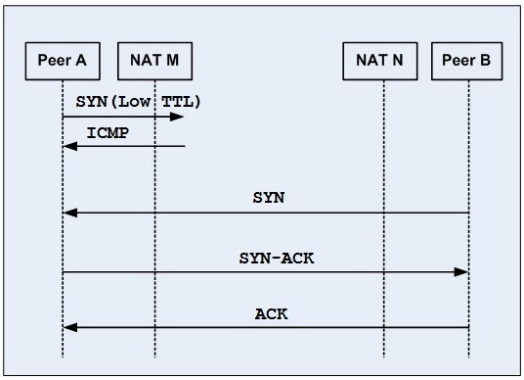



Figure 2.16: STUNT#2.

This approach has four significant problems. Firstly, it requires the peer to determine a TTL that is large enough to cross its own NATs but low enough not to reach the other peer's NAT. The problem is that when the two outermost NATs share a common interface, such a TTL does not exist. The second problem is that the ICMP TTL - exceeded error could be generated in response to the SYN packet and this might be interpreted by the NAT as a fatal error. Thirdly, the TCP sequence number of the initial SYN might be changed when it arrives at the NAT. The fourth problem is that it depends on a third-party to spoof a packet for an arbitrary address.

In the second approach, which is known as STUNT#2, it is also required that the STUNT server, which is omitted in Figure 2.16, discovers the public IP address and port number of the NAT and the NAT type. Unlike STUNT#1, the STUNT server does not need to spoof the SYN-ACK packet and therefore, it does not need to have the root or administrator privilege. Only one peer sends out a low-TTL SYN packet and then terminates the connection attempt and creates a TCP socket (passive) on the same address and port. The other peer then initiates a regular TCP connection, as shown in Figure 2.16. As with STUNT#1, it is important for the peer to pick an appropriate TTL value and the NAT must not consider the ICMP error to be a fatal error. It also requires that the NAT accepts an inbound SYN following an outbound SYN which is a sequence of packets not normally seen.

The main contributions of the STUNT are to predict the next port number and establish the direct TCP connections.

#### 2.5.1.5 *NATBlaster*

*NATBlaster* [5] is another approach similar to STUNT#1 but avoids the IP spoofing requirement. In this approach, both peers initiate an outbound connection and keep the record of their initial sequence number. The two peers then interchange the sequence numbers and each peer sends a SYN-ACK packet to another. The SYN-ACK packet is sent into the network using

a RAW socket. Once the SYN-ACKs are received by both peers, ACKs packets are sent to complete the connection setup. Figure 2.17 shows the packet flow in NATBlaster.

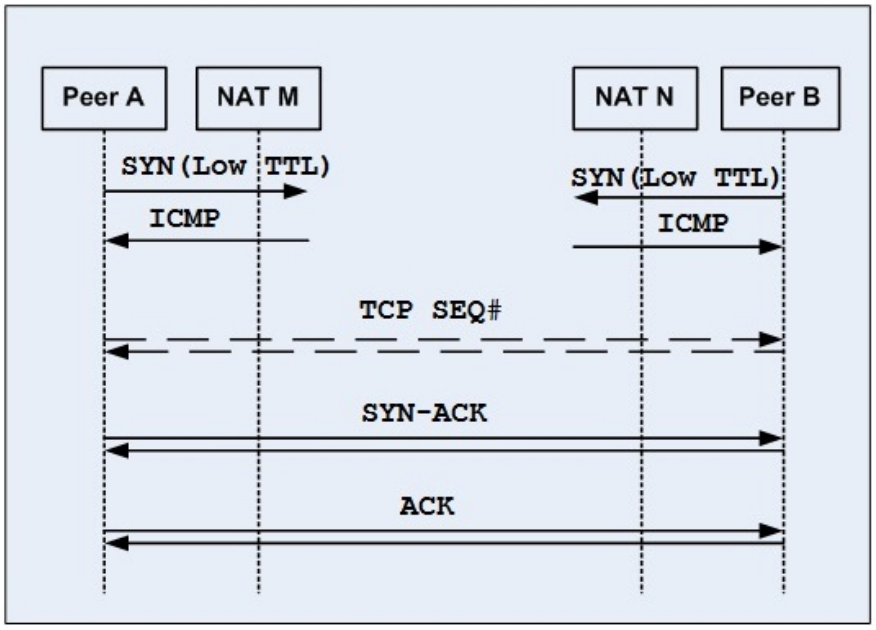



Figure 2.17: NATBlaster.

Like the STUNT#1 approach, NATBlaster also requires the peer to properly select the TTL value, requires the NAT to ignore any ICMP error and it fails if the NAT changes the sequence number of the SYN packet. In addition, it requires the client applications to have access to RAW sockets, which are usually available at root or administrative privilege levels.

#### 2.5.1.6 *Peer-to-Peer NAT*

The mechanism of *Peer-to-Peer NAT* [25] is similar to STUNT#2. In Peer-to-Peer NAT, both peers send SYN packets and listen for any incoming connections to the same port at the same time. One of the NATs will end up following the simultaneous open sequence, where the other one follows the regular open sequence. Figure 2.18 shows the packet flow in Peer-to-Peer NAT. Basically this approach is not as popular as STUNT#2 and it additionally requires the peer to retry failed connection attempts until a time-out occurs.

Table 2.1 presents a brief summary of all the NAT traversal techniques along with their advantages and disadvantages described here. In the second column of the table, 'Cone' means full cone, restricted cone and port restricted cone; as described in Section 2.5.

### 2.5.2 *Impact of Routing Scheme over NAT*

Selecting a proper routing algorithm is another crucial factor to design a P2P overlay. The routing algorithm defines how the information will be routed through the P2P network.

Table 2.1: A summary of NAT traversal techniques.

| **NAT traversal techniques** | **NAT types** | **UDP /TCP** | **Advantages** | **Limitations** |
|---|---|---|---|---|
| **STUN** | Cone | UDP | 1) Does not require any changes on NAT devices. Clients can learn NAT devices automatically. 2) Easy and standardised. | 1) Short term solution. 2) Requires client application to be upgraded to support STUN and an additional STUN server residing in the public Internet. |
| **UDP Hole Punching** | Cone | UDP | 1) Preserves the transparency of NAT 2) Works even with multiple levels of NATs. | 1) By introducing a server, it turns the part of a P2P network into Client-Server model. 2) It adds overhead to the bandwidth and increases communication latency as well. |
| **TCP Hole Punching** | Cone | Both | Provides a direct TCP connection which is a reliable connection with error detection and packet retransmission | Additional processes required to establish the TCP handshake as well as to synchronise the packet sequence numbers. |
| **TURN** | All types | Both | 1) TURN can traverse every type of NAT including symmetric NATs. 2) TURN is used to relay media streaming between peers. | 1) TCP data is proxyied by a third party which creates a potential network bottleneck. 2) Requires excessive network resource requirements 3) TURN server must remain available for the whole duration of the allocation. 4) Unlike STUN, TURN does not allow a direct connectivity between NATed hosts. |
| **STUNT**#1 | All types | TCP | Direct TCP connection | 1) Need supervisor privileges. 2) Determining a proper TTL value. 3) Requires IP address spoofing. |
| **STUNT**#2 | All types | TCP | 1) Direct TCP connection. 2) No need for administrator privilege. | Determining a proper TTL value. |
| **NATBlaster** | All types | TCP | Does not require IP Spoofing. | 1) Need supervisor privileges. 2) Determining a proper TTL value. |
| P2P**-NAT** | All types | TCP | Easy. | 1) It is not as popular as STUNT#2. 2) Packet flood may occur. |

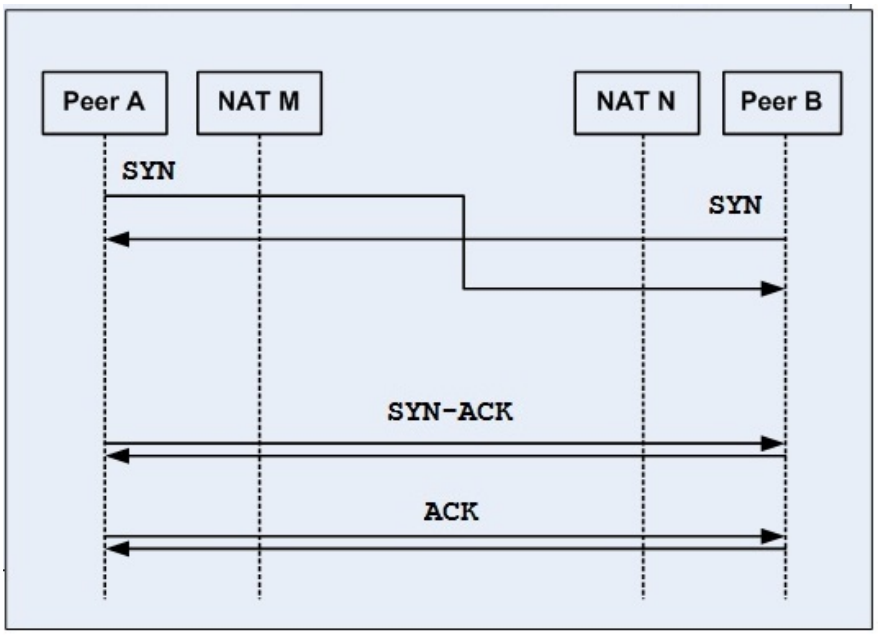



Figure 2.18: P2P-NAT.

There are two common routing schemes among existing DHT based P2P overlays: Iterative Routing and Recursive Routing. Furthermore, the recursive routing has four other variants: Full-Recursive, Semi-Recursive, Hybrid and Full-Hybrid Routing.

#### 2.5.2.1 *Iterative routing*

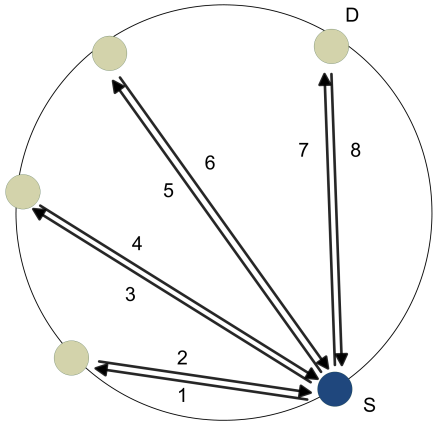
In *Iterative Routing*, the initiating node sends a request to an intermediary node which replies directly to the initiator node with a nearer location. The initiator then sends a new request to the nearer recommended node and the lookup process continues until the final target node is reached.

#### 2.5.2.2 *Recursive Routing*

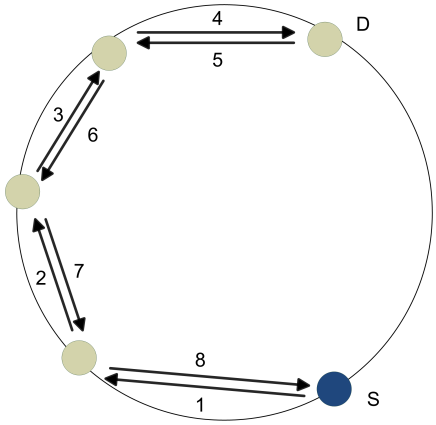
In *Recursive Routing*, the initiator sends the request to an intermediary node. If this intermediary node does not have the resource, it forwards this request to another node that it believes to be the nearest to the destination. This continues until the request arrives at the destination node. Then the destination responds back to the initiator using a reverse path that the request has traversed to reach the destination.

#### 2.5.2.3 *Full-Recursive Routing*

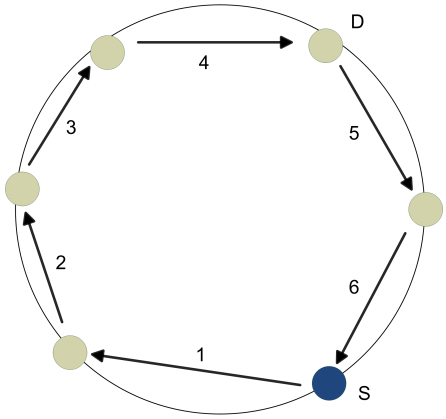
The Full-Recursive routing is similar to recursive routing, except the response is routed recursively back to the originator using its own routing table to which a connection already exists.



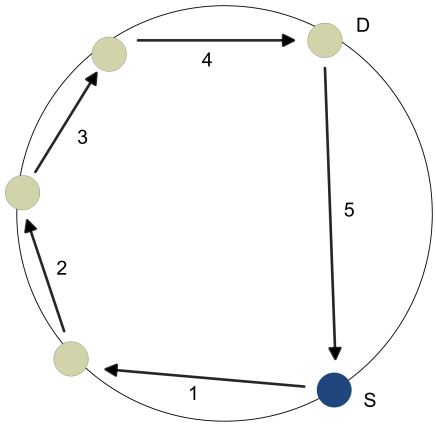

(a) Iterative.



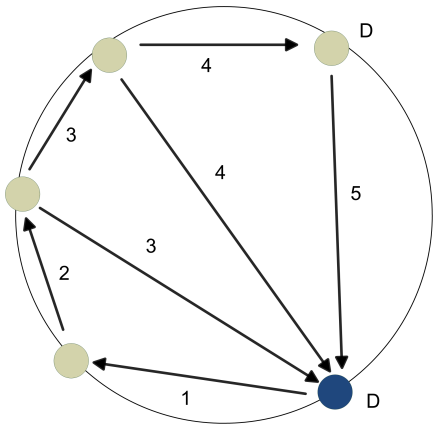

(b) Recursive.



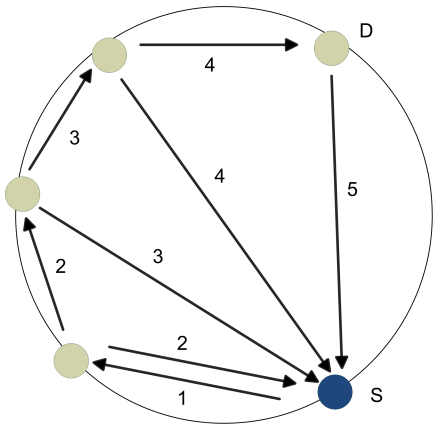

(c) Full-Recursive.




(d) Semi-Recursive.




(e) Hybrid.




(f) Full-Hybrid.

Figure 2.19: Different possible routing methods.

#### 2.5.2.4 *Semi-Recursive Routing*

In Semi-Recursive Routing, the lookup query is routed in the same way as recursive routing, however, the way the response is routed back is different. Here, once the destination is reached, the response is directly sent back to the initiator.

#### 2.5.2.5 *Hybrid Routing*

Hybrid routing is a combination of iterative and recursive routings. It is basically a recursive routing with an additional step where at every hop, a node sends an acknowledgement back to the initiating node except the first neighbouring contacted node. Therefore, the initiating node constantly receives an update about the path [46].

#### 2.5.2.6 *Full-Hybrid Routing or Direct Routing*

This approach is similar to hybrid routing. The only difference is that the acknowledgement is also received from the first contacted node.

Figure 2.19 shows each routing scheme.

#### 2.5.2.7 *Discussion*

Researchers argue that recursive routing is more efficient due to the reduced number of total messages transmitted, which eliminates the corresponding latency between sending the messages and obtaining a response [10]. In addition, it is more resilient to IP connectivity restrictions and advantageous when dealing with NAT.

On the contrary, iterative routing is heavily affected when a querying node cannot communicate directly with some nodes on the path due to NATs.

Semi-recursive routing arguably has the smallest lookup hop count as well as a smaller lookup latency than the recursive and iterative routings. As it follows the similar technique of recursive scheme, it is also advantageous over NAT.

Both Hybrid and Full-Hybrid routings have the advantage of the lower delay of recursive routing and the faster failure recovery of iterative routing.

Table 2.2 presents the total number of messages passed in a lookup query involving $i$ nodes [10].

Table 2.2: Comparison of different routing methods.

| **Routing Scheme** | **Maximum possible number of messages passed** | **Total number of additional NAT holes to be punched** |
|---|---|---|
| **Iterative** | $2(i-1)$ | $i-1$ |
| **Recursive** | $2(i-1)$ | $0$ |
| **Full-Recursive** | $i$ | $0$ |
| **Semi-Recursive** | $i$ | $1$ |
| **Hybrid** | $2(i-1)-1$ | $i-1$ |
| **Full-Hybrid** | $2(i-1)$ | $i-1$ |

As NATs prohibit a direct communication between nodes, those nodes behind NATs have to rely on some other techniques such as STUN, TURN or the hole punching mechanism in order to communicate with each other. For the above discussed routing schemes, NATs have different implications on their efficiencies.

For example, in the hole punching technique, a hole (or a connection) must be established between the communicating nodes, and that the hole also needs to be maintained by exchanging keep-alive messages in some cases.

Considering every node is behind a NAT, Table 2.2 shows the total number of new NAT hole punches that must be made by all nodes for the lookup query. It is considered that during periodic DHT maintenance, the connections to immediate neighbours have already been established and therefore, it is not included in the count. It is also considered that the first query is always sent to a neighbour (such as Chord).

In recursive routing, a node establishes a hole with only neighbouring nodes and has no need to establish new holes on the way as responses are routed back through the same connections. Similarly in full-recursive routing, the response of the destination is routed back to the originator through the known nodes, there is no need to punch any extra hole.

The semi-recursive routing is similar to recursive routing, except when the resource is found, a new connection (or a hole) needs to be opened between the source and destination node. This extra cost of hole-punching is included in the cost of the semi-recursive routing.

In the iterative routing, each hop needs to establish a new connection. The originator node sends the first lookup query to its direct neighbour and all subsequent connections are with non-neighbours. Therefore, $i - 1$ new NAT connections (holes) must be established, where $i$ is the total number of querying nodes in the network.

The hybrid and full-hybrid routing are the combination of iterative and recursive routing and, each hop needs to establish a new connection except that the first neighbour will not send any reply/acknowledgement to the originator. Therefore, $i - 1$ new NAT holes must be punched.

Evidently in a NATed environment, the recursive and full-recursive routing eliminate the number of new connection that must be established through the NATs to allow communication over other routing schemes. Again, in the recursive (and full-recursive) and hybrid (and full-hybrid) routing networks, the intermediate nodes send and receive more messages than the source and destination node. In a communication network such as mobile network where the nodes have limited processing power it is important to reduce the burden of processing power.

### 2.5.3 *Analysis of Structured P2P Overlays with NAT*

There are a very small number of attempts to apply NAT Traversal techniques in structured P2P overlays. This section will analyse two of them.

#### 2.5.3.1 *SMBR*

*SMBR (Selective-Message Buddy Relaying)* [77] uses a server-less distributed NAT traversal mechanism for DHT based P2P overlays. DHT algorithms have two kinds of messages: control messages and data messages. Control messages are small sized packets used to search and publish the resource file, whereas data messages are big sized packets used to transfer the files. SMBR uses different techniques for traversing control messages and data messages. At first, the *Private Nodes (PNs)*, which are behind NATs randomly select buddies, which are *Global Nodes(GNs)* that have public IP addresses. Once the PNs have found their buddies, they can traverse the control messages via the buddies' relay. For traversing the data messages, a direct connection is made with the help of the buddies. If the communication has to be made in between two PNs, the UDP hole punching technique is used. In this case, the traversal time is higher than relaying which is used to traverse the control messages. The authors claim that this is cost efficient as the two nodes send and receive a lot of data after traversal. Again, the load of the buddies is only for sending and receiving a small number of packets and processing these packets, which is much lighter than relay. Therefore, the system can achieve a balance between the traversal time and the buddy's load.

The authors implemented this technique in a prototype system of Kademlia and tested this approach in a smaller LAN environment. There is no evaluation regarding the performance of the overlay. It has not been evaluated how the SMBR will perform, if the network has high churn. The authors also claimed that the approach could be used in Chord with minor changes but they did not describe how it could be possible. Furthermore, the selection process of buddies is random and does not ensure that it will reach the destination node in a shorter distance.

#### 2.5.3.2 *MaidSafe*

The MaidSafe Platform [51] is a distributed P2P network that allows individuals and companies to create secure, efficient and fast applications to manipulate all kinds of data, share applications, data and communications between individuals and/or groups. It uses a P2P system and replaces existing client-server centric systems. It utilises Kademlia-based routing and the routing performs the hole punching technique to enable direct communication between each pair of nodes in combination with the MaidSafe-RUDP which is an implementation of reliable UDP. Hole punching can be achieved even without the requirement for any servers. But this technique works as long as both communicating nodes are not behind any symmetric

NAT. If both nodes are behind symmetric routers, the relay option is used to traverse the NAT. Fairness is an issue in this case where the relay node has to provide more bandwidth to the NATed node.

## 2.6 EXISTING P2P SOLUTIONS IN MOBILE NETWORKS

In this section, various approaches to implement P2P overlays in wireless cellular networks are presented and discussed. The approaches discussed here are based on structured P2P overlays and the deployed systems are not simply using off-the-shelf DHT algorithms, instead they use novel designs directly addressing mobile network issues.

### 2.6.1 *Hybrid Chord Protocol (HCP)*

The *Hybrid Chord Protocol (HCP)* [80, 82] establishes a structured P2P network to cope more efficiently in mobile environments. It is based on the Chord Protocol [74]. The HCP is structured by addressing two different nodes, *Static* nodes and *Temporary* nodes. Static nodes are highly available nodes and have high data rate connections to other static nodes as well as large CPU and storage capabilities. On the other hand, the temporary nodes are low-performance nodes that join the network only for a short period of time. The routing tasks are performed by all nodes in the network, however, storing and providing the shared object references are processed only by the static nodes. Therefore, the availability of shared contents increases. Temporary nodes store only a pointer to their closest static successor. Thus, if a temporary node receives a request for an assigned key, it simply forwards the request to its closest static successor which stores the corresponding references for this key.

The authors have shown that the routing and maintenance traffic in the overlay can be reduced significantly as the object references are shifted only when static nodes, which normally have long session times, join or leave the network.

The main issue of this protocol is the high dependency on static peers. In the evaluation, the authors have shown that 40% of the nodes have to be static in the HCP network [82]. On the other hand, if the mean online time of temporary peers such as PDA or mobile peers decreases from 10 minutes to 2 minutes then the failure probability increases from 35% to 50% which clearly shows that the system is not efficient in high churn scenarios such as in mobile networks. Most importantly, mobile networks come with the NAT functionalities, which are not covered in this system.

### 2.6.2 *Mobile Node ID Kademlia Architecture*

Huiyou *et al.* [78] introduced the *Mobile Node ID Kademlia Architecture* for transmitting multimedia streams in wireless networks. In this system, the key space is divided into two parts: the *main ID* and the *assistant ID*; where the main IDs are distributed to the stable nodes and the assistant IDs are considered for unstable nodes such as mobile phones. The main ID nodes take part in DHT for locating resources whereas the assistant ID nodes communicate only with the main node. When an assistant node wants to get a resource, it must contact the corresponding main node first and then it can calculate the location through another hash table on the main node. The authors further designed the *Virtual Channel Caching Mechanism* and the *Motion Estimated Caching Mechanism* to guarantee the continuity of the multimedia data.

In this system, if a node joins/leaves frequently, only the direct main nodes are affected without causing any harm to the whole system. The authors argued that their approach would work efficiently in large networks and allow fluent multimedia streaming.

The drawback of this architecture is that the main ID nodes have to be very robust and stable nodes with a number of properties such as large network bandwidth, computing ability and the connective ratio. The architecture has been implemented in a P2P simulator considering a very small network of 1,000 nodes. The simulated performance is compared only with Kademlia. This system is also aimed for mobile networks; however, it does not deal with high churn and NAT traversal.

### 2.6.3 *Chordella*

*Chordella* [37] is a two-tier P2P overlay network system designed for mobile networks. The system architecture defines two different classes of peers: *Super-Peers* and *Leaf-Nodes*. The upper level peers are Super-peers that are participating devices with substantial resources and have a more reliable network connection. They establish a ring-shaped structured DHT overlay based on Chord. A Super-peer connects to the Leaf-nodes and acts as a proxy to allow them to participate in the overlay. The low level nodes are Leaf-nodes which are resource-constrained wireless devices such as mobile phones. They do not provide high data rates, nor do they require high computational power or even a certain uptime for their participation in the network. As the Leaf-nodes only maintain an overlay connection with their Super-peers, they need to be able to recognise and react to a Super-peer failure. To solve this, they periodically run a simple PING/PONG algorithm. They also store a list of other available Super-peers in the system to be able to rejoin the overlay network on the failure of a Super-peer.

In addition to the basic DHT features, a *Load Balancing Algorithm* [83] was implemented to ensure the same load levels of the participating Super-peers. Another *Optimal Operation Point*

*Selection Algorithm* [81, 84] was designed to adjust the number of Super-peers dynamically to a cost-optimal value. An integrated *Caching Algorithm* was implemented to improve the system performance and reliability by adaptively caching references to popular contents.

The main focus of this system is on cost optimisation and load balancing between the participating super-peers. However, during the churn phase, which is set for 2 hours in the simulation, the Chordella shows worse load balancing performance [83]. Again, the system does not deal with NAT traversal.

### 2.6.4 *Cellular Chord (C-Chord)*

The *Cellular Chord (C-Chord)* P2P system [85] integrates the cellular users into the well-known Chord P2P system in a topology-aware fashion. The C-Chord provides the cellular users a choice of downloading the contents either from the Internet peers at a faster rate or from the other cellular peers within the same base station, avoiding the Internet data penalty. The model consists of one main Chord (*m*-Chord) ring that contains Internet peers, and several auxiliary Chord (*a*-Chord) rings equal to the number of base stations containing all cellular peers. *a*-Chord rings are connected to the *m*-Chord ring with a *special key*. Every base station has a unique Cell-ID and cellular users retrieve this Cell-ID through a cell search procedure. An *m* bit identifier, denoted as $\mathrm{id}_{cell}$, is generated by hashing each Cell-ID. The data associated with this identifier $\mathrm{id}_{cell}$ is a list of *J* number of IP addresses of cellular users in the *a*-Chord under the same base station and has the format of $IP^{key} = \{ip_1, ip_2, ....., ip_J\}$. The cellular peer in the *a*-Chord ring maintains a gateway called $IP^{gw}$ which is a list of $\{ip_1, ip_2, ....., ip_{J'}\}$ containing the IP addresses of the *successor* and *predecessors* of the special key. For efficient lookup, the *m*-Chord ring and *a*-Chord rings also maintain *finger tables*. However, each cellular peer maintains a finger table with fewer entries, as the number of peers in each *a*-Chord is less than the number of Internet peers in the *m*-Chord; therefore, the system reduces the high management cost of routing information.

The peer selection module is based on the *Stable Marriage (SM) Algorithm* [27], which chooses the appropriate candidate from the discovered potential senders. However, this algorithm requires a lot of messages (i.e. $O(n^2 - n + 1)$ messages, where $n$ is the number of requesting peers) and it becomes unrealistic when the number of requesting peers or potential senders is large. The authors proposed an alternative *Heuristic Peer Selection* scheme which is near-optimal and requires fewer messages (i.e. $O(n)$ messages).

The system does not measure the degree of churn and fault tolerance. Again, NAT traversal has not been considered in this design.

### 2.6.5 *Bushfire*

*Bushfire* [7] is a framework based on a structured P2P overlay that operates in mobile networks and supports developers to create third party applications. It consists of two main parts: a handset component to interface with the third party applications and a proxy server, called SysProxy, to overcome the constraints imposed by NAT and the mobile network operators. Kademlia DHT is implemented on top of the star topology network. The Kademlia implementation and the NAT traversal functionality together work as a framework.

The proposed Bushfire framework utilised a 'proof of concept' software application, referred to as *BUTE (Budapest University of Technology & Economics)* [42]. The main difference between BUTE and Bushfire is that the BUTE application works using only WiFi capability, in the range of a WLAN access point; whereas Bushfire provides the connection of the P2P nodes into the public GSM mobile telephone network using appropriate mobile handset API functionality. Again, BUTE does not support the NAT traversal capability; whereas Bushfire gains full Internet access and overlay connectivity through the mobile operator network using SysProxy.

However, placing a SysProxy server in the design is a significant bottleneck for data flow and the system resiliency. Another limitation of Bushfire is that a maximum $14,535$ users can be supported in the application. Node churn and data longevity within the overlay are two other key factors that have not been considered in Bushfire.

### 2.6.6 *Discussion*

From the above discussions, it is clear that a number of different structured P2P approaches exist that can potentially be used for large scale applications in mobile networks. The different characteristics of the presented systems make it difficult to compare and contrast them directly against each other. Table 2.3 presents different approaches according to the main goal of the architecture, P2P overlay protocol, overlay structure, lookup process, overlay maintenance, performance evaluation, churn handling, proposed applicability and prototype implementation.

It is interesting to analyse that the main design goal of the maximum approaches presented here is to perform the efficient lookup and to reduce the maintenance overheads, thereby improving stability in mobile networks.

The structured P2P solutions compared here are mainly based on either Chord or Kademlia.

Using the concept of a superpeer is a good way to handle different issues regarding the power consumption of resource constrained mobile devices with short battery lives. In a hierarchical P2P network, mobile peers could collaborate with a static/fixed peer with high

Table 2.3: Comparison of existing mobile P2P.

| **Protocols** | **HCP** | **Mobile Node ID Kademlia** | **Chordella** | **C-Chord** | **Bushfire** |
|---|---|---|---|---|---|
| **Main goal** | To perform faster lookup and increase the stability of the structure. | To minimise the huge maintenance overheads by introducing the heterogeneity of participating nodes. | To provide efficient resource lookup in highly heterogeneous environments using diverse devices. | To provide cellular users an effective P2P platform in a topology-aware fashion. | To provide a framework to support developers to create 3rd party applications. |
| **P2P Overlay Protocol** | Chord | Kademlia | Chord | Chord | Kademlia |
| **Overlay Structure** | Ring | Tree | Ring | Ring | Star |
| **Lookup Process** | When a temporary node receives a query, it simply forwards the request to its closest static node. | For Lookup, the requesting node gets the main ID first; then it reads its hash table for the desired node. | When a Leaf-node asks for a resource, the directly connected Super-peer resolves the lookup by determining the responsible Super-peer based on the object's key. | The cellular peer sends a query to its own a-Chord successor or to any of the nodes from the list IPgw. | SysProxy forwards the ping message to the appropriate device. Receiving the response; the FindValue routine takes further actions. |
| **Overlay Maintenance** | Uses Token-like stabilisation where two or more tokens circulate in counter-clockwise direction on the ring. | Three different operations can take place according to different situations: Replace-when there is a better main node; Split-when the current main node has too many assistant nodes; Combine-when there is no new candidate. | Super-peers periodically run Chord's original STABILISE and a slightly modified FIX-FINGER algorithms. Furthermore, the Super-peers refresh all the references periodically in order to keep them up-to-date. | Both m-Chord and a-Chord maintain finger tables and with less entries in the cellular peer's finger table, the system reduces the management cost of routing information. | All high level logical activities related with routing, DHT maintenance and data storage or retrieval from the network are managed by ping, findNode, findValue and store messages. |

| Protocols | HCP | Mobile Node ID Kademlia | Chordella | C-Chord | Bushfire |
|---|---|---|---|---|---|
| **Performance evaluation** | It reduces the maintenance traffic by a factor of $1/\alpha$ compared to Chord; where $\alpha$ is the ratio of the mean online time of static nodes in HCP and the mean online time of all nodes in Chord. | The system performs better than standard Kademlia when the resource file is big and the number of nodes in the network is large. | It ensures the same load level of the participating Super-peers using a Load Balancing Algorithm. Also dynamically adjusts the number of Super-peers to a cost-optimal value and improves the system performance and reliability by adaptively caching references to popular content. | It improves the performances with smaller hop-count per lookup query compared to traditional Chord. | Energy optimisation experiments reveal that the mobile device consumed less power when the routing table refresh frequency is longer. |
| **Churn handling** | Handles a moderate level of churn. | Does not handle churn. | Considers a churn phase upto 2 hours but performs poorly. | Does not handle churn. | Does not handle churn. |
| **NAT Traversal** | Not available. | Not available. | Not available. | Not available. | Implemented NAT traversal using a proxy server. |
| **Proposed Applicability** | File sharing. | Multimedia Streaming. | Content sharing, Person tracking, Dynamic services. | File sharing. | Any kind of 3rd party apps. |

computational power for basic operations such as resource discovery or maintaining the connectivity of the mobile peer through the mobile network. In this way, those static or superpeer nodes can provide the mobile peer the capability of full interaction between the other peers in the network. HCP, Chordella and C-Chord are particularly adapted in that way to implement the P2P approach in a mobile network; however, each approach has different routing and design techniques. The mobile node ID Kademlia architecture also allows the

system to have stable and unstable nodes where the stable nodes have the larger network bandwidth and only they can participate in the DHT system for resource discovery.

Each approach performs better for some specific criteria; however, it is not mentioned how they will perform in other criteria. For example, HCP reduces the maintenance traffic, but how the participating static peers in the network balance the load is not stated. Again, C-Chord minimises the hop-count, but the authors of the protocol have not mentioned how to tackle fault-tolerance of the system.

HCP and Chordella have been measured under a moderate level of churn, but both of them have performed poorly. The rest of the approaches have not considered churn in their systems.

Unlike others, Bushfire is a software development framework to incorporate P2P applications and can handle the NAT/Firewall functionality using a proxy server.

HCP and C-Chord are designed mainly for file sharing applications, the Mobile Node ID Kademlia approach is applicable for media streaming whereas Chordella is applicable for content sharing, person tracking and dynamic services.

## 2.7 REVIEW OF P2P SIMULATORS

A large number of simulations have been performed throughout the thesis to evaluate the performance of DHT based P2P overlays using a P2P network simulator. There are a number of active P2P simulators available. To select a P2P simulator with the desired functionalities, a detailed review of the existing P2P simulators has been carried out. A summary of these tools is provided in Table 2.4.

A brief discussion of each simulator is presented below.

PeerSim [54] is a Java based simulator whose main focus is to provide high scalability, with network sizes of up to $10^6$ nodes. However, this scalability comes at the cost of omitting the behaviour of the underlying communication network.

P2PSim [30] is a discrete event simulator for P2P overlays written in C++. Models for Chord, Accordion [23], Koorde [39], Kelips [33], Tapestry [79], and Kademlia are available. However, these implementations are specific to P2PSim and do not model all features of the protocols. P2PSim has been simulated with up to $3,000$ nodes using the Chord implementation. This simulator is largely undocumented and therefore hard to extend.

Overlay Weaver [70] is a toolkit for P2P Overlays written in Java. It has been tested with tens of thousands of nodes (their website quotes $300,000$). Chord, Kademlia, Pastry, Tapestry and Koorde models are available. The simulations have to be run in real-time environments and there is no statistical output which limits its use.

PlanetSim [28] is a discrete event simulation framework for both structured and unstructured overlays, written in Java. It has a modular, well-structured architecture and services can be re-used for other overlays. Chord and Symphony [52] models exist and can consist of up to

Table 2.4: A comparison of available active P2P simulators.

| *Simulator* | P2P *Protocols* | *Network Size* | *Language* |
|---|---|---|---|
| PeerSim | Collection of internally developed P2P models | $>10^6$ | Java |
| P2PSim | Chord, Accordion, Koorde, Kelips, Tapestry, Kademlia | 3,000 | C++ |
| Overlay Weaver | Chord, Kademlia, Koorde, Kelips, Tapestry and FRT-Chord | Tens of thousand | Java |
| PlanetSim | Chord, Symphony | 100,000 | Java |
| NS-2 | Gnutella | Not known | C++/OTel |
| NS-3 | CAN, Pastry | Not known | C++, Python |
| SSFNet | Chord, EpiChord | 33,000 | Java/C++ |
| OverSim | Chord, Kademlia, Broose, Pastry, Bamboo, Gia | 100,000 | C++ |
| PeerfactSim.Kom | CAN, Chord, Kademlia, Gia, C-DHT, Pastry | 50,000 | Java |
| D-P2P-Sim+ | Chord | 400,000 | Java |

100,000 nodes. However, it provides rather limited support to collect statistics. It has a much simplified underlying network layer without consideration of bandwidth and latency costs.

NS-2 [38] is a discrete-event simulator that provides substantial support for simulation of lower layer protocols. Only one P2P protocol, Gnutella, is available in NS-2. Simulations in NS-2 are built using C++ and OTcl. It is mostly used with small networks and due to the models' complexity is generally unsuitable for large scale P2P networks.

NS-3 [57] is a general-purpose and discrete-event simulation framework that provides models for packet data networks and provides a simulation engine to conduct simulation experiments. It is not a backwards-compatible extension of NS-2, but a new simulator. It is constructed using C++ and Python with scripting capability. There is no official P2P model released in NS-3, however, contributors modelled CAN and Pastry using NS-3.

SSFNet [17] is a discrete-event simulation framework written in Java and C++. This framework is built on the *Scalable Simulation Framework (SSF)* and uses the *Domain Modelling Language (DML)* to configure networks. Chord and EpiChord have been implemented in SSFNet. There is a claim that SSFNet manages to run models with 33,000 nodes, however, the authors of the original EpiChord paper [47] could not simulate networks with more than 10,000 nodes.

OverSim [3] is an open-source P2P simulation framework for the OMNeT++ simulation environment. It provides a generic lookup mechanism and an *RPC (Remote Procedure Calls)* interface to facilitate additional protocol implementations. It allows large-scale simulations of simplified networks as well as complex heterogeneous underlay networks. Several P2P algorithms such as Chord, Kademlia, Bamboo [64], Broose, Koorde [39], NICE [4], NTree [29], Pastry, and GIA [14] have been implemented in OverSim. Models can scale to over $100,000$ nodes.

PeerfactSim.Kom [73] is a discrete event based P2P simulator environment. Its focus is on being extensible and on large scale network models. This simulator offers the potential to model different types of P2P systems including distributed *Content Delivery Networks (CDNs)*, streaming applications and overlay systems. It comes with a built-in churn generator. The simulator includes models of lower layers but does not yet include TCP.

D-P2P-Sim+ [71] is a distributed simulation environment which employs multi-threading, asynchronous message passing and a distributed environment with a graphical user interface. There is little information on this simulator besides a short paper and a poster. These report simulated network sizes of up to $400,000$ nodes. It seems that the only implemented overlay in this simulator is Chord. The system is extensible and other algorithms could be implemented. Multiple computers running the simulator may be interconnected to achieve larger simulated network sizes.

Based on this study OverSim has been selected for the experimentation due to its flexibility with respect to underlay characteristics, possible high scalability and strong support for DHT-based P2P algorithms. OverSim also provides an interactive GUI and real-time messages which are extremely useful for debugging models. Other surveys of P2P network simulators can be found in research by Brown and Kolberg [9], and Naicken *et al.* [56].

### 2.7.1 *OverSim*

OverSim [3] is a modular simulation framework that uses *Discrete Event Simulation (DES)* to simulate exchange and processing of network messages. The layered architecture of OverSim consists of an *Application layer*, an *Overlay layer* and an *Underlying Network* which is illustrated in Figure 2.20. The Common API provides the communication between overlay and application layers [18]. Each overlay protocol that wants to use this API has to provide at least a *Key-Based Routing (KBR)* interface to the application. The application layer can further be divided into two sub-layers: Tier 1 and Tier 2. Simple applications can be implemented in the Tier 1 layer, whereas complex applications are programmed in the Tier 2 layer.

To facilitate the implementation of multiple overlays, OverSim supports several functions such as an overlay message handler, generic lookup function and global observer. The message handler provides a *Remote Procedure Calls (RPC)* interface to deal with time-outs and packet

retransmission due to packet losses. It also collects message related to statistical data. The lookup function supports a generic iterative and recursive lookup interface. The Global Observer offers a global view of the overlay network in every simulation.

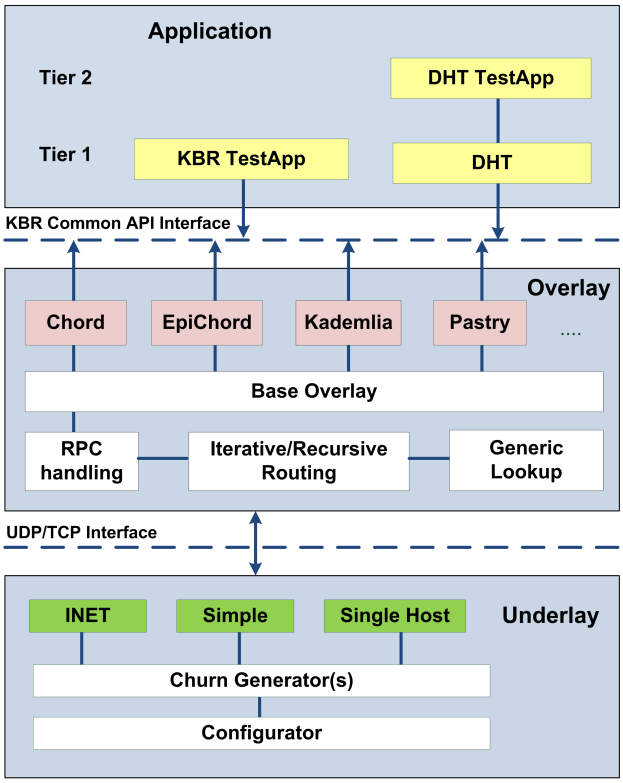



Figure 2.20: Layered architecture of OverSim: application layer, overlay layer and underlay layer.

The underlying network in OverSim is abstracted using an underlay model as shown in Figure 2.20. OverSim supports different models of underlay, namely: Simple Underlay, INET Underlay and SingleHost Underlay. The Simple Underlay is the default underlay model in OverSim where nodes are added within a logical access network in a 2-dimensional Euclidean space. With its low computational overhead and highly accurate performance, this model is suitable for large scale simulation. The SingleHost Underlay emulates a single host within an overlay. It provides a middleware facility to allow OverSim nodes to communicate with real networks. The INET Underlay is based on the INET framework which offers much more flexibility in simulating different network topologies and facilitates extensibility to develop additional functionalities. The framework itself is based on OMNeT++ and utilises the same concept where modules can communicate with each other by message passing. The framework also provides a full-stack implementation of different link-layer models such as PPP, Ethernet and 802.11, and several other network protocols such as IPv4, IPv6, TCP, SCTP and UDP.

This chapter provided a brief background on P2P overlays especially on DHT-based structured P2P overlays. In addition, it discussed related work on existing NAT traversal techniques and P2P solutions regarding mobile networks.

Section 2.2 defined the P2P overlay with a discussion of its two categories: unstructured and structured P2P overlays. The unstructured P2P overlay does not follow any particular structure; peers can be organised using a random data structure, whereas the structured P2P overlay strictly follows a tightly controlled data structure such as Ring, Mesh or Tree structure and provides a unique mapping between the nodes and the data objects using distributed hash tables. This type of P2P is much more efficient than unstructured P2P especially on data lookup. Therefore, this thesis focuses solely on structured P2P overlay networks. Next, Section 2.3 described five popular structured P2P overlays which will be evaluated and experimented in the next chapter to identify the most suitable configuration of these overlays in order to achieve the optimal performance in mobile environments.

Section 2.4 provided a list of challenges to adopt P2P in mobile and wireless networks. Among the challenges mentioned, this thesis focuses on three particular issues: high churn rate, data consumption and NAT restriction. Next, a review of some efforts undertaken to solve the churn problem were presented. The researchers have identified some design considerations and methods to deal with churn. A review of some works relevant to the energy and battery consumption of mobile devices was presented as well. These works reveal that developing a P2P overlay in mobile devices is feasible, however, making the system energy efficient is still very difficult.

Section 2.5 presented a brief overview of NAT and NAT terminologies. To make a proper P2P connection and overcome the NAT barrier over mobile networks, a suitable NAT traversal technique is necessary. This section also explained a number of available techniques for NAT traversal. STUN and UDP hole punching are for UDP traffic, and STUNT, NATBlaster, P2P NAT are for TCP traffic, whereas, TURN and TCP hole punching are for both UDP and TCP traffic. All the techniques bring some benefits as well as have some drawbacks. Almost all techniques employ a central server or a rendezvous server to solve the NAT issue. However, the hole punching is the most robust and practical NAT traversal technique. This section also provided a discussion regarding the impact of routing schemes over NAT. The iterative routing is heavily affected when a querying node cannot communicate directly with some nodes on the path due to NAT, whereas the recursive routing eliminates the number of new connections that must be established through the NATs to allow communication with the iterative routing scheme. Two new techniques (SMBR and MaidSafe platform) were discussed which had been implemented to provide server-less distributed NAT traversal mechanisms for DHT-based P2P overlays.

Finally, Section 2.6 provided a number of novel approaches to implement P2P overlays in wireless cellular networks. The existing mobile P2P overlays were analysed. It has been found that only a small number of structured P2P solutions are available on cellular networks and there is not a single approach that solves the churn, energy consumption and NAT problems. Again, only a very small number of approaches have been implemented in real systems. Substantial efforts addressing these issues are still required to adapt DHT-based P2P overlays to the characteristics of mobile networks.

Throughout the thesis, all the experiments have been carried out using a P2P simulator, OverSim. Before choosing OverSim, it was necessary to review available P2P simulators. Section 2.7 presented a review of a number of available P2P simulators and justified why OverSim was selected for the experimentations.

The next chapter will start with the evaluation of a number of structured P2P overlays and identify the most suitable overlay to be used in mobile networks based on high churn and bandwidth consumption.

# Part II

# CONTRIBUTIONS

# 3

# PERFORMANCE EVALUATION OF STRUCTURED P2P OVERLAYS UNDER CHURN

*The previous chapter provided background information on* P2P*, the issues regarding deploying* P2P *in mobile networks and NAT traversal. From the related work, it is found that there is no* P2P *approach that solves the problem of high churn, bandwidth consumption and NAT issues. This chapter investigates churn and bandwidth consumption issues further by evaluating the performance and efficiency of a number of popular DHT based structured* P2P *overlays.*

## 3.1 INTRODUCTION

We have witnessed a tremendous growth of P2P in high speed wired networks. Recently there has been a trend in exploring the use of P2P applications in wireless mobile networks. Moving from high powered desktop computers to hand-held mobile devices with limited battery power and restricted resources highlights the importance of performance and efficiency of P2P.

Therefore, it is important to investigate DHT based P2P overlays considering the essential requirements to enable P2P in mobile networks and identify the most important parameters of DHTs that affect their performance. The overlays also need to be compared under different conditions of mobile networks such as churn and bandwidth consumption. A series of simulations have been used to evaluate the performance and efficiency of five popular DHT based structured P2P overlays: Chord, Pastry, Kademlia, Broose and EpiChord, taking into account the conditions as seen in mobile networks.

The rest of this chapter is organised as follows. Section 3.2 briefly summarises the key requirements to deploy P2P from the perspective of mobile networks. Section 3.3 describes the experimental methodology used in this research. Then in Section 3.4, a number of DHT based structured P2Ps have been evaluated to find the most suited parameters with respect to the features identified in Section 3.3. Section 3.5 evaluates the performance and efficiency of the overlays. Finally, Section 3.6 briefly summarises the chapter.

## 3.2 KEY REQUIREMENTS

Before starting the experimentation, a set of key features that can be used to compare Chord, Pastry, Kademlia, Broose and EpiChord must be identified in order to determine the suitability

of these DHTs in mobile networks. A set of features suitable for application scenarios running on mobile networks was proposed in [35]. Below, the key requirements are briefly summarised.

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

## 3.3 EXPERIMENTAL METHODOLOGY

### 3.3.1 *Simulation Platform*

For all experiments throughout the thesis, the OverSim [3] platform, implemented in C++ and developed on top of the OMNeT++ framework, has been used, due to its strong support of DHT based P2P algorithms. It provides a generic lookup mechanism and an RPC interface to facilitate additional protocol implementations. It allows large-scale simulations of simplified networks as well as complex heterogeneous underlay networks. It also provides an interactive GUI and real-time messages which are extremely useful for debugging. OverSim has been used in a number of other P2P works, making the results easily comparable to other researchers' works [2].

### 3.3.2 *Performance Metrics*

The following *performance metrics* are evaluated from the simulation results:

- ***Lookup Success Ratio:*** The percentage of successful lookups in the overlay.
- ***Lookup Hop Count:*** The mean number of overlay hops to send a message from a source to a destination node.

- ***Mean Maintenance Traffic Load:*** The mean number of maintenance bytes sent per second by each node. This metric will be considered as a parameter for bandwidth consumption in this thesis.

### 3.3.3 *Simulation Setup*

Throughout the evaluation and experimentation discussed in this chapter, the following setup has been used, unless stated otherwise.

- ***Network Size:*** Experiments have been carried out using a $10,000$ node network.
- ***Churn:*** The level of churn is simulated by various node lifetime values between 100 seconds and $1,000$ seconds. A shorter lifetime means a higher level of churn. The node lifetime of 100 seconds to simulate high churn scenarios has been selected according to the study presented in [36].
- ***Churn Model:*** The simulations employ OverSim's *Lifetime Churn model* with a *Weibull distribution* consisting of scale parameter, $a$ and shape parameter, $b$. Here, $a$ is calculated in the following way:

  $$a = \frac{\mathtt{LifetimeMean}}{\mathtt{tgamma}(1+\frac{1}{b})}$$

  Where $\mathtt{LifetimeMean}$ is the average node lifetime, $\mathtt{tgamma}$ function returns the gamma function of $(1 + \frac{1}{b})$ and $b = 1.0$.

  The Weibull distribution has been shown to model churn in P2P overlays much more accurately [75].
- ***Lookup Interval:*** Each node issues lookups for random keys at intervals exponentially distributed with a mean of 60 seconds. Unless otherwise stated, all simulations are carried out using this lookup interval.
- ***Transport Protocol:*** The selected P2P overlays for the experimentations use only UDP as a transport protocol, except Chord. Thus, UDP has been used in the evaluations.
- ***Routing:*** As iterative routing works better in high churn environment [76], the overlays have been configured to use *Iterative* routing.
- ***Repetition:*** Each configuration has been repeated 5 times, and results have been averaged. In the plots, the 95% confidence intervals have been calculated across the repetitions, to increase the credibility of the simulation results [61].

The overall evaluation process includes two simulation steps. During the first step, the values of key parameters for each overlay are determined in order to yield the best possible performance under high churn. Thus, this step fine tunes the parameters to yield an optimal configuration for each overlay. For this step, the performance has been investigated at 100 seconds and 1,000 seconds of average node lifetime. In the second step, the best performing configurations of all overlays are compared against each other to identify the best performing overlay under churn.

Next, the first step of simulation results of Chord, Pastry, Kademlia, Broose and EpiChord are analysed over a range of tunable parameters and their values.

### 3.4.1 *Chord*

For Chord, Table 3.1 lists the key parameters, which are *stabilisation delay*, *fixfinger delay*, *successor list size*, *check predecessor delay* and *extended finger table size*, along with the varied range of values. Figure 3.1 to 3.5 show the simulation results for *success ratio* and *bandwidth consumption* with varying values for these simulation parameters.

Table 3.1: Simulation parameters for Chord.

| **P2P Protocol** | **Parameters** | **First step** | **Second step** |
|---|---|---|---|
| *Chord* | Stabilise Delay (s) | 5, 10, 20, 30, 40, 50, 60, 120 | 20 |
| | Fixfinger Delay (s) | 30, 60, 90, 120, 180, 240, 300 | 30 |
| | Successor List Size (nodes) | 4, 8, 16, 32 | 8 |
| | Check Predecessor Delay (s) | 5, 10, 30, 60 | 5 |
| | Size of Extended Finger Table | 0, 1, 4, 8, 16 | 0 |

In Chord, each node learns about newly joined nodes and updates its successors and predecessor after a stabilisation delay. For step 1, the stabilisation delay has been varied between 5 seconds and 120 seconds. Figure 3.1 shows that a low stabilisation delay improves the success ratio but also increases the bandwidth consumption. Therefore, a value of 20 seconds has been chosen as stabilise delay.

In Chord, finger (routing) tables are updated in fixed intervals (fixfinger delay). The fixfinger interval was varied between 30 and 300 seconds. Figure 3.2 shows that a lower value improves the success ratio. A value of 30 seconds has been selected to get the best possible lookup success ratio at a moderate level of bandwidth usage.

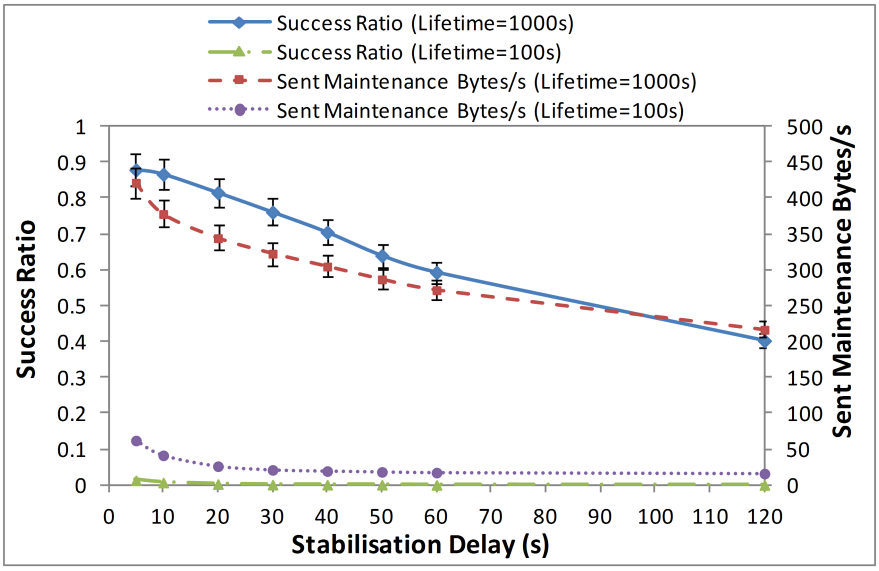



Figure 3.1: Chord: Testing on stabilisation delay.

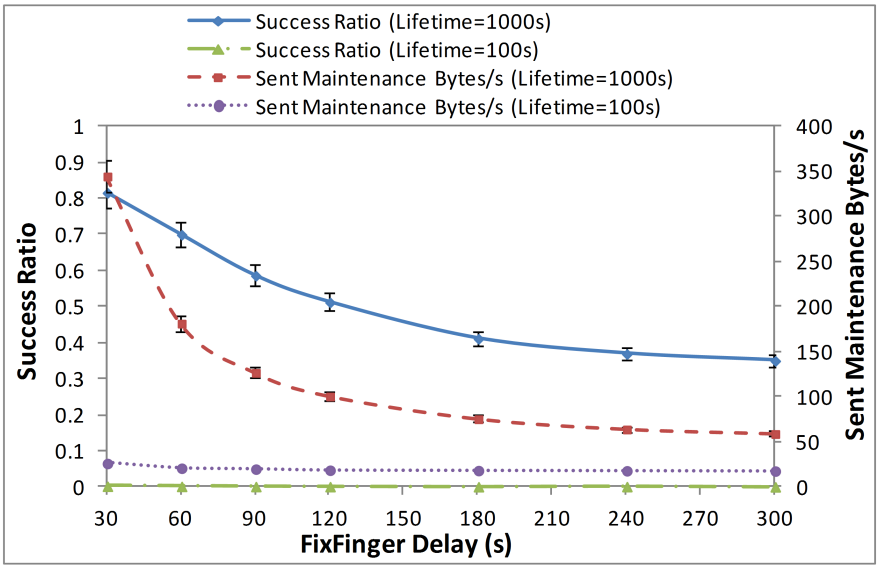



Figure 3.2: Chord: Testing on fixfinger delays.

Results of varying the successor list size and the check predecessor delay are depicted in Figure 3.3 and 3.4 respectively. A successor list size of 8 and the check predecessor delay of 5 seconds have been chosen to maintain a reasonable success ratio while minimising the maintenance traffic.

Figure 3.5 shows the effect of using extended finger tables. Based on these results, the size of the extended finger table has not been altered; as it was found that any increase leads to more traffic with no significant improvement in lookup performance.

Overall, the very poor lookup performance of Chord at high churn should be noted. This will be revisited in the second simulation step when the different overlays will be compared against each other.

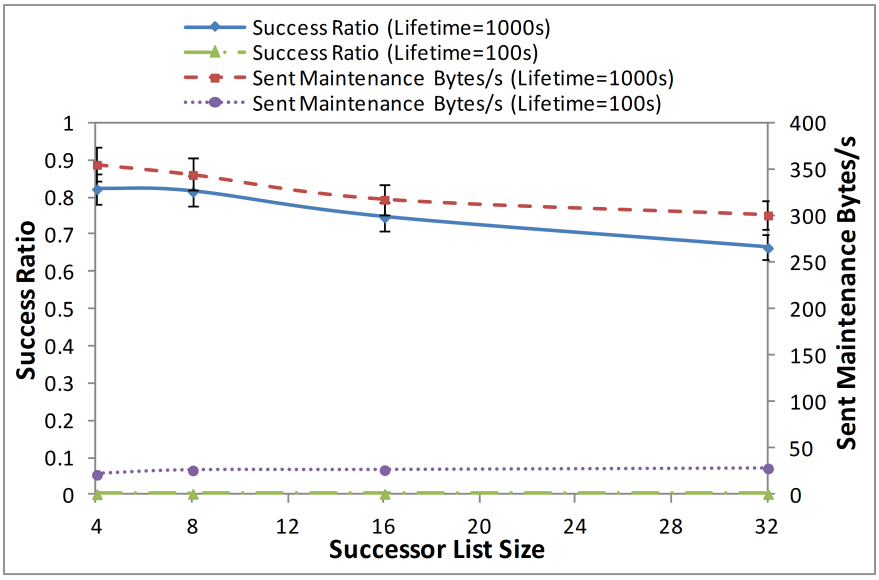



Figure 3.3: Chord: Testing on successor list sizes.

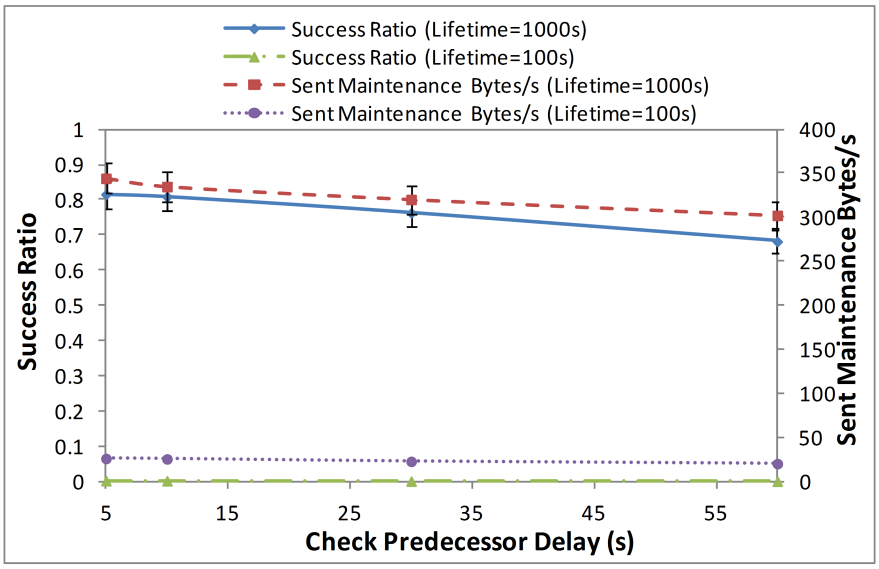



Figure 3.4: Chord: Testing on check predecessor delays.

### 3.4.2 *Pastry*

The key parameters of Pastry are *Leaf set size*, *Neighbourhood set size*, *Bits per digit* and *lookup redundant nodes*, as shown in Table 3.2. Figure 3.6 to 3.10 show the simulation results while varying these parameters.

In Pastry, each node maintains a *Leaf* set and a *Neighbourhood* set. A leaf set contains the number of leaves or nodes $L$, with the $L/2$ numerically closest smaller and $L/2$ numerically closest larger nodeIds from the present node's nodeId. It is similar to Chord's *successor* or *predecessor* list. Figure 3.6 shows the effect of different numbers of leaf nodes. In the low churn scenario ($1,000$ seconds), there is no significant lookup performance difference. In the high

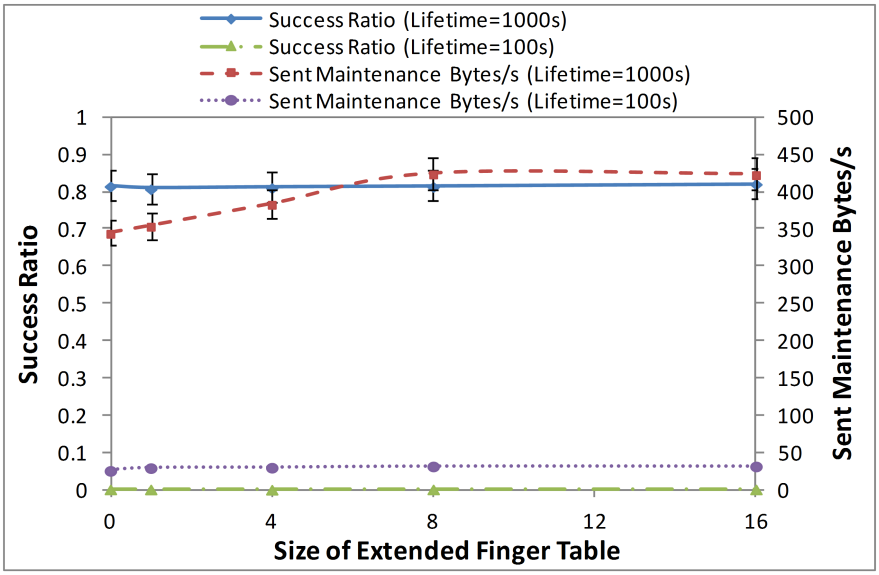



Figure 3.5: Chord: Testing on extended finger table sizes.

Table 3.2: Simulation parameters for Pastry.

| P2P Protocol | Parameters | First step | Second step |
|---|---|---|---|
| *Pastry* | Leaf Set Size (nodes) | 4, 8, 16, 32 | 8 |
| | Bits per Digit (bits) | 1, 2, 4, 6 | 4 |
| | Neighbourhood Set Size (nodes) | 0, 2, 4, 8, 16 | 0 |
| | Lookup Redundant Nodes (nodes) | 2, 4, 8 | 4 |

churn scenario (100 seconds) the highest success ratio has been observed when using 8 leaf nodes. Hence, the number of leaf nodes has been selected as 8.

The results of varying levels of bits per digit are shown in Figure 3.7. Using a higher number of bits can improve the *success ratio* as well as *hop count* (Figure 3.8) but it increases the maintenance cost. Considering the results in the high churn scenario, 4 bits per digit have been selected.

A Neighbourhood set contains nearby nodes based on proximity metrics. In Figure 3.9, Pastry has been simulated *with neighbourhood sets* and *without neighbourhood sets* and no considerable improvements have been observed in the success ratio (and bandwidth consumption). Therefore, it has been decided not to use any neighbourhood set.

In Figure 3.10, the effects of altering the number of redundant nodes are shown. Higher values of redundant nodes can improve the success ratio but also cause an increased amount of maintenance traffic. Therefore, a fixed value of 4 has been chosen to balance between a good lookup success ratio and maintenance traffic load.

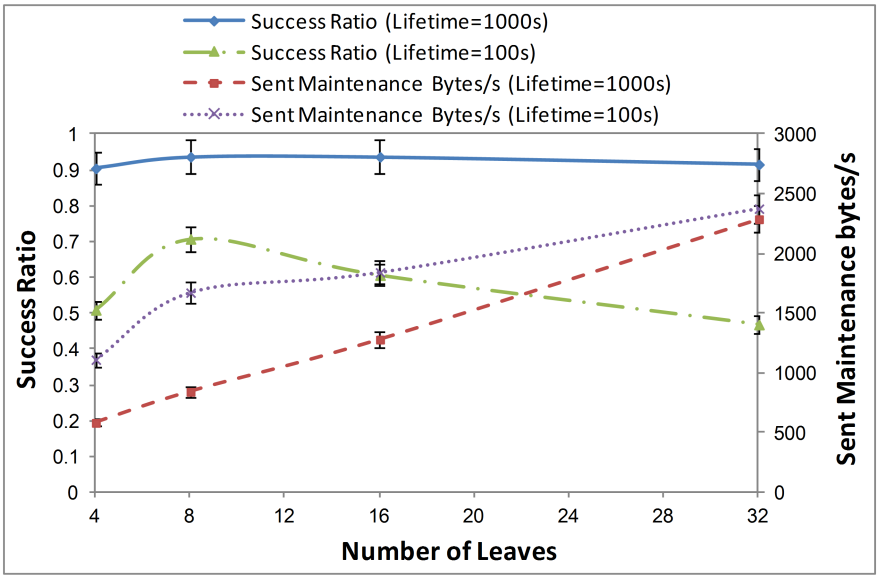



Figure 3.6: Pastry: Testing on leaf set.

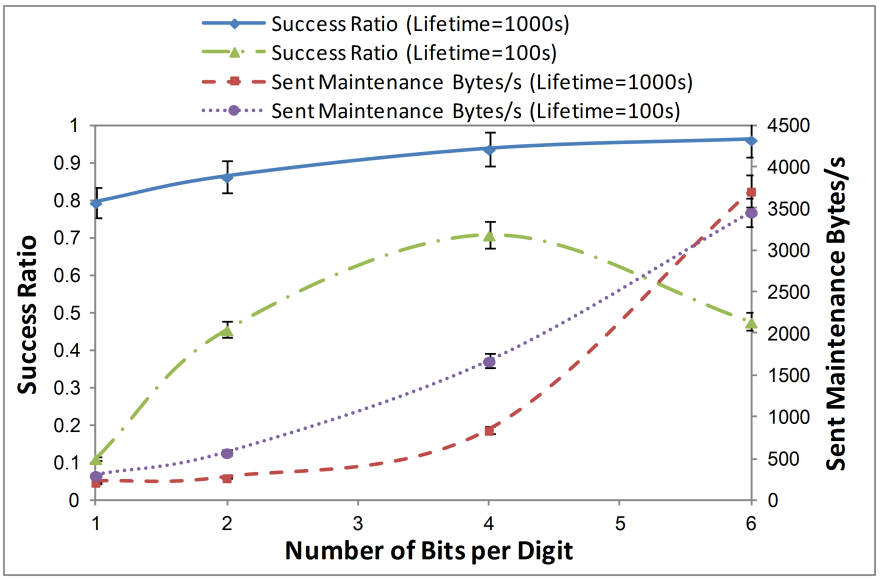



Figure 3.7: Pastry: Testing on bits per digits(i).

### 3.4.3 *Kademlia*

Kademlia has been experimented with its key parameters: *bucket size (k)*, *siblings (s)*, *redundant nodes (r)*, *bits per digit (b)*, *bucket refresh interval* and *parallel lookups*, as shown in Table 3.3. Figure 3.11 to 3.16 illustrate the *success ratio* and *bandwidth consumption* for different parameters of Kademlia.

Each node in Kademlia maintains a routing table that consists of k buckets. Each bucket also holds a fixed number, k, of references to reach other nodes. After simulating different

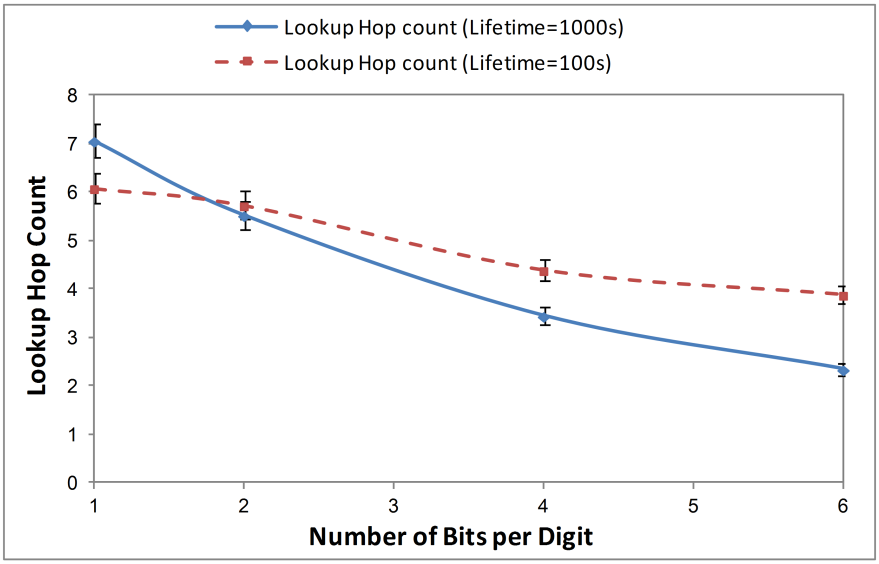



Figure 3.8: Pastry: Testing on bits per digits(ii).

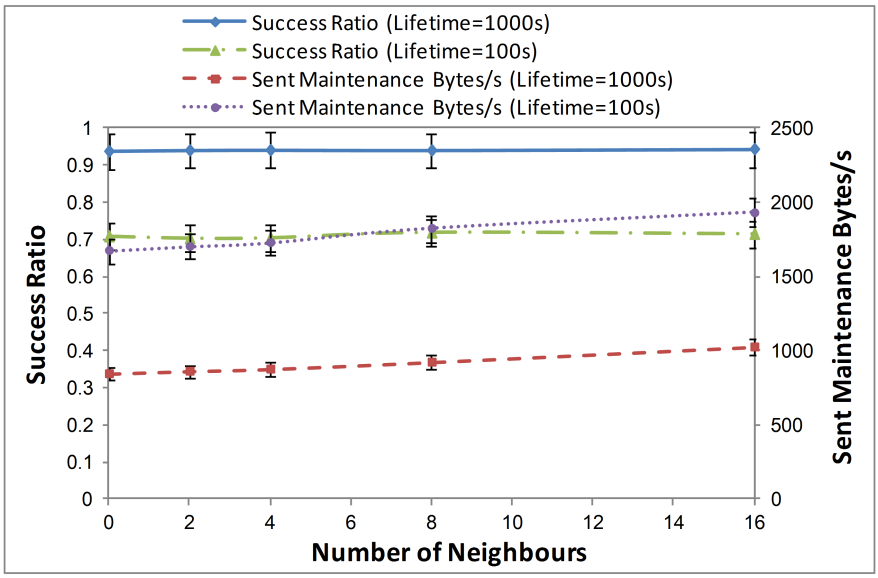



Figure 3.9: Pastry: Testing on neighbourhood set.

values for k, 8 has been selected as the optimal value according to the success ratio and sent maintenance bytes (Figure 3.11).

Siblings are those nodes that are responsible for a certain (key, value) pair which are needed to be stored in a DHT. Comparing different values in Figure 3.12, it has been found that 2 sibling nodes are sufficient to achieve the best performance in high churn scenarios.

To achieve the best performance, the trade-off between a higher success ratio and the decreased maintenance traffic has been evaluated, and therefore, the value of 8 for redundant nodes and 1 for bit per digit have been chosen from the simulation results as shown in Figure 3.13 and Figure 3.14 respectively.

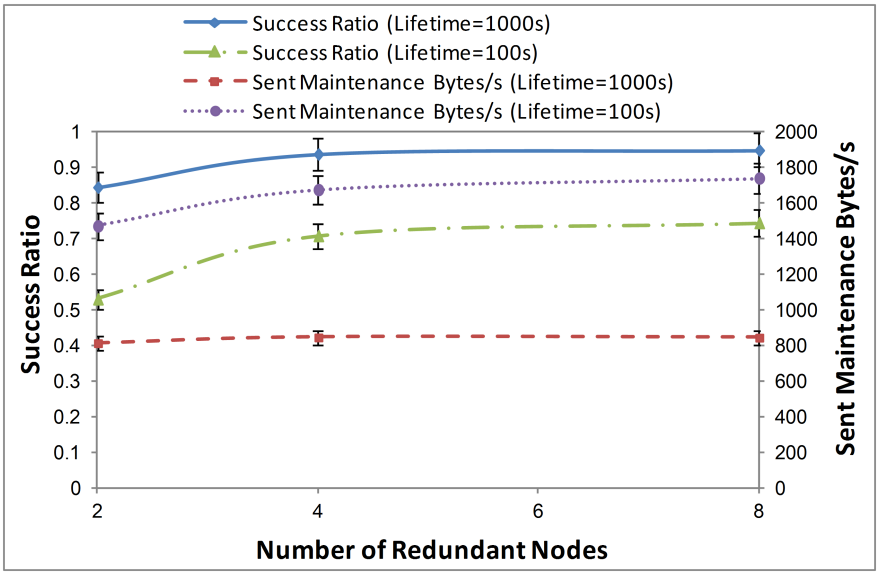



Figure 3.10: Pastry: Testing on redundant nodes.

Table 3.3: Simulation parameters for Kademlia.

| P2P Protocol | Parameters | First step | Second step |
|---|---|---|---|
| *Kademlia* | Bucket Size, k | 4, 8, 16, 32, 40, 64, 72, 80 | 8 |
| | Siblings, s (nodes) | 2, 4, 8, 16, 32 | 2 |
| | Lookup Redundant Nodes, r (nodes) | 1, 2, 4, 8, 16, 20 | 8 |
| | Bits per Digit, b (bits) | 1, 2, 4, 6, 8 | 1 |
| | Bucket Refresh Interval (s) | 100, 600, 1000, 3000, 5000 | 1000 |
| | No. of Parallel Lookups | 1, 2, 3, 4, 5 | 3 |

In Figure 3.15, the bucket refresh interval has been altered. As shown in the figure, an interval of 1,000 seconds is sufficient to keep the buckets consistent.

As Kademlia supports parallel lookups, the effect of parallel lookups has been investigated in Figure 3.16. In high churn environments, parallelism can increase the success ratio significantly. Therefore, a value of 3 has been selected for lookup parallelism which shows a moderate increase in bandwidth.

### 3.4.4 *Broose*

Table 3.4 lists the parameters of Broose along with the experimented values.

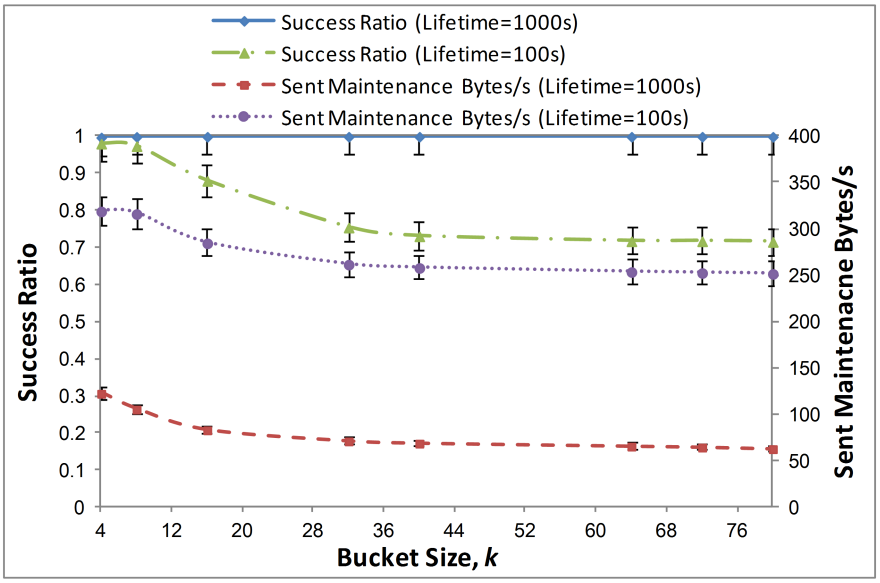



Figure 3.11: Kademlia: Testing on bucket size, k.

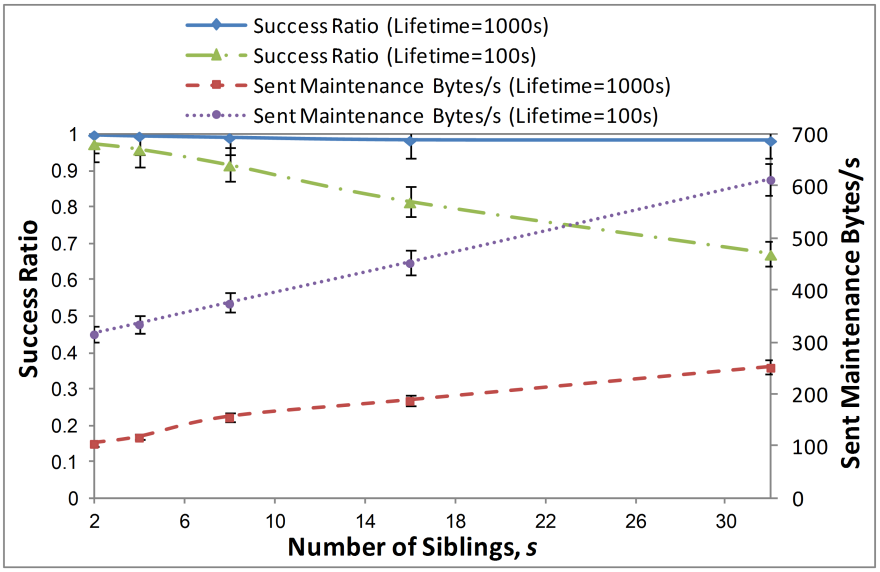



Figure 3.12: Kademlia: Testing on siblings, s.

Table 3.4: Simulation parameters for Broose.

| P2P Protocol | Parameters | First step | Second step |
|---|---|---|---|
| *Broose* | Bucket Size, k | 4, 8, 16, 32 | 8 |
| | Bucket Refresh Interval (s) | 30, 60, 120, 180, 300, 600 | 30 |
| | Shifting Bits (bits) | 2, 3, 4 | 2 |
| | No. of Parallel Lookups | 1, 2, 3, 4, 5 | 3 |

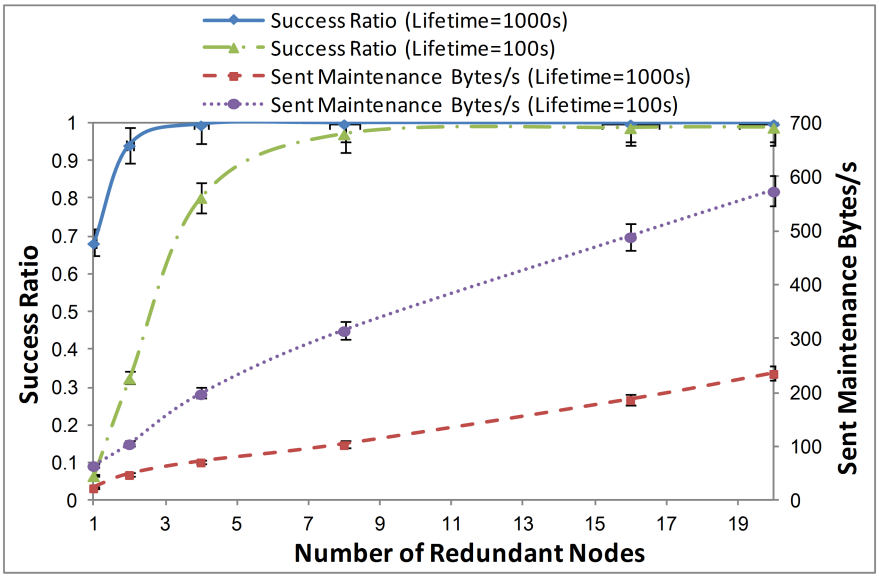



Figure 3.13: Kademlia: Testing on redundant nodes, r.

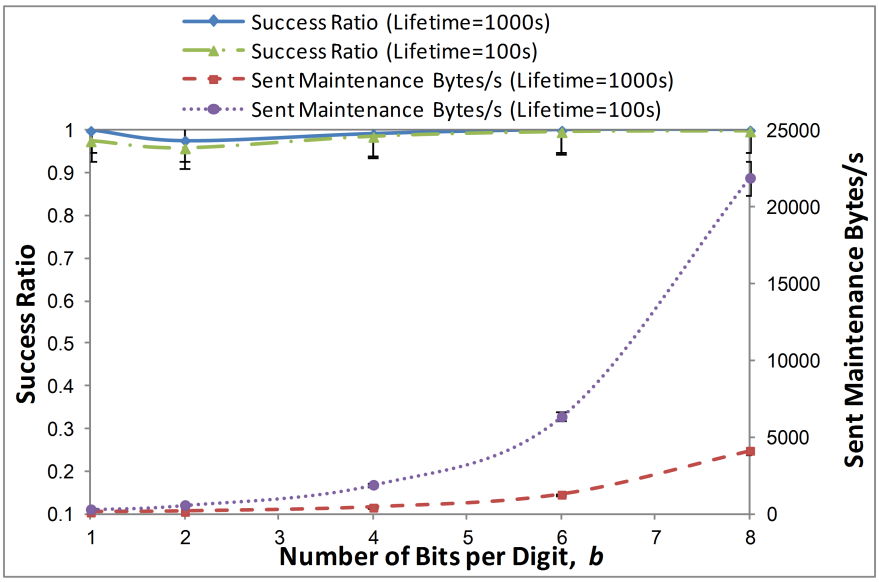



Figure 3.14: Kademlia: Testing on bits per digits, b.

The bucket size $k$, and bucket refresh interval have been varied in Figure 3.17 and 3.18 respectively. Fig 3.17 shows that the *success ratio* is improved with larger buckets; however, it also increases the *maintenance traffic* drastically. Therefore, a value of 8 for $k$ is appropriate. In terms of the bucket refresh interval, a value of 30 seconds has been selected in order to get the best performance as well as a moderate level of maintenance traffic under high churn (Figure 3.18). Shifting more than one bit at each routing step clearly improves the success ratio, however, it also increases the maintenance traffic (Figure 3.19). Thus, a value of 2 bits has been selected. Increasing parallel lookups does not show any significant lookup performance improvement (Figure 3.20). Hence, the value of 3 has been selected.

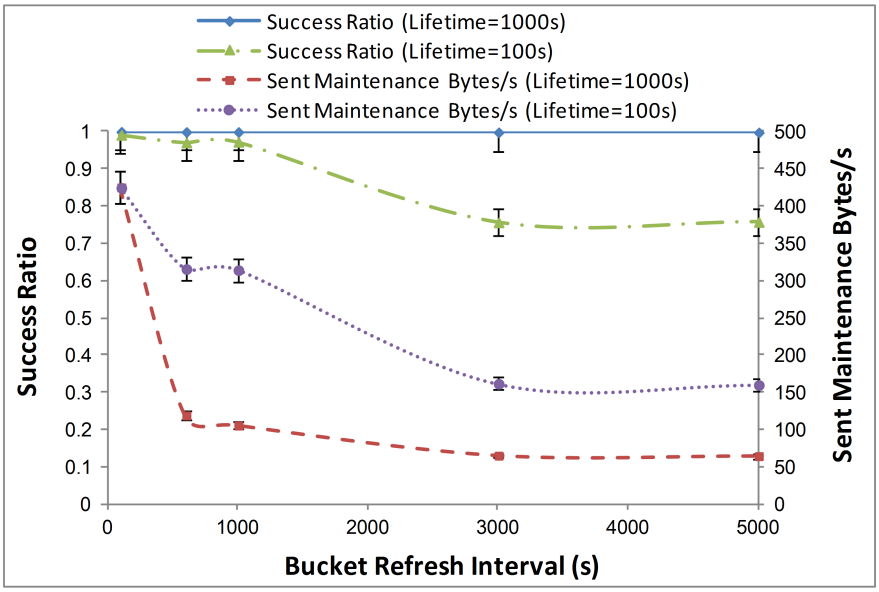



Figure 3.15: Kademlia: Testing on bucket refresh interval.

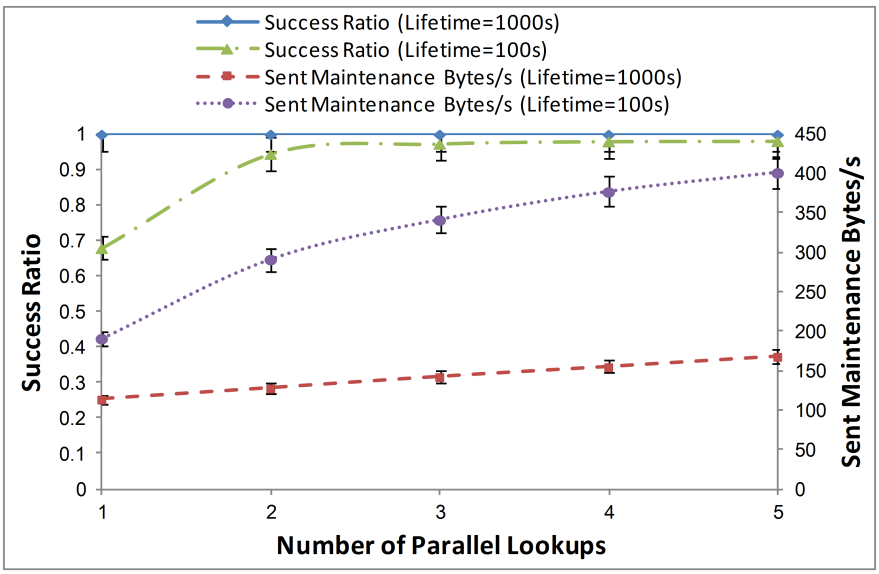



Figure 3.16: Kademlia: Testing on parallel lookups.

### 3.4.5 *EpiChord*

Table 3.5 lists the parameters of EpiChord which are: *stabilise delay*, *successor list size* and *parallel lookups*. Figure 3.21 to 3.24 show the results of *success ratio* and *bandwidth consumption* for EpiChord.

The stabilisation delay and successor list size have been altered in Figure 3.21 and 3.22 respectively. Increasing the stabilisation delay does not affect the performance of EpiChord regarding the success ratio and bandwidth consumption; therefore, a value of 60 seconds has been selected. The successor list size of 4 is a fair compromise between the performance of

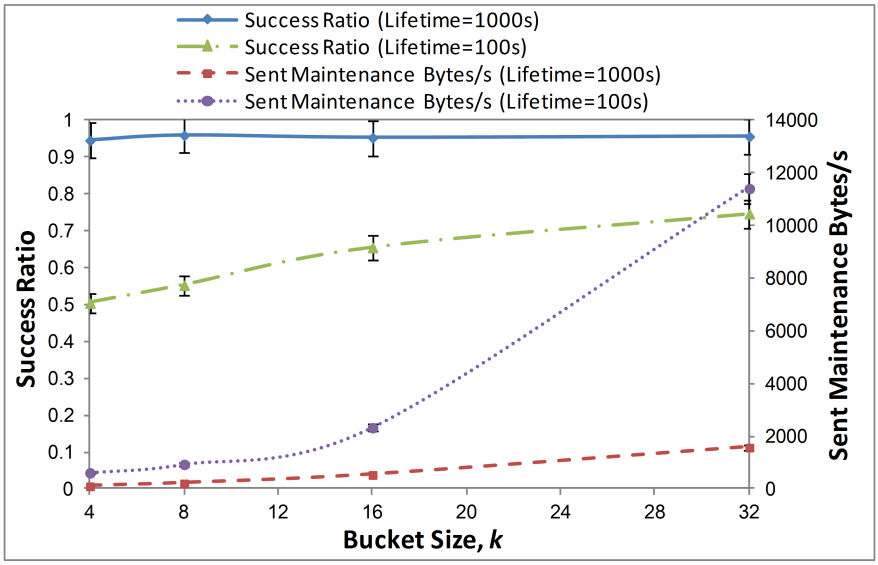



Figure 3.17: Broose: Testing on bucket size, k.

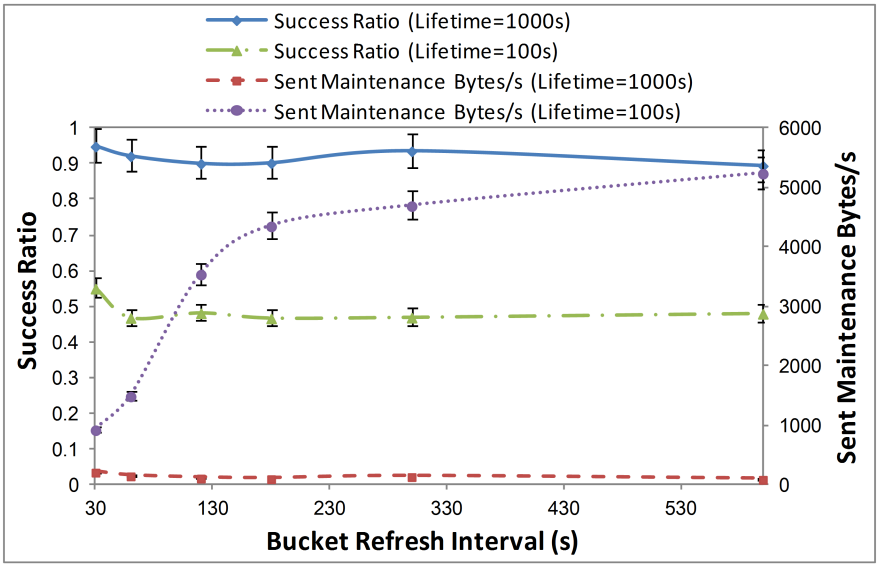



Figure 3.18: Broose: Testing on bucket refresh interval.

Table 3.5: Simulation parameters for EpiChord.

| P2P Protocol | Parameters | First step | Second step |
|---|---|---|---|
| *EpiChord* | Stabilise Delay (s) | 10, 20, 60, 100 | 60 |
| | Successor List Size (nodes) | 2, 4, 8 | 4 |
| | No. of Parallel Lookups | 1, 2, 3, 4, 5 | 3 |

success ratio and maintenance traffic load. Figure 3.23 shows the effect of parallel lookups on success ratio and maintenance traffic. However, in EpiChord, parallel lookups do not

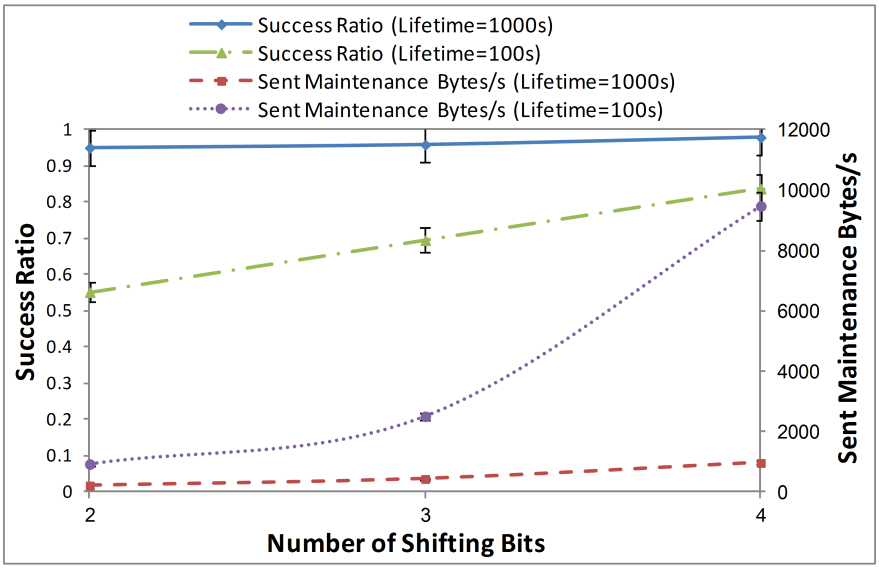



Figure 3.19: Broose: Testing on shifting bits.

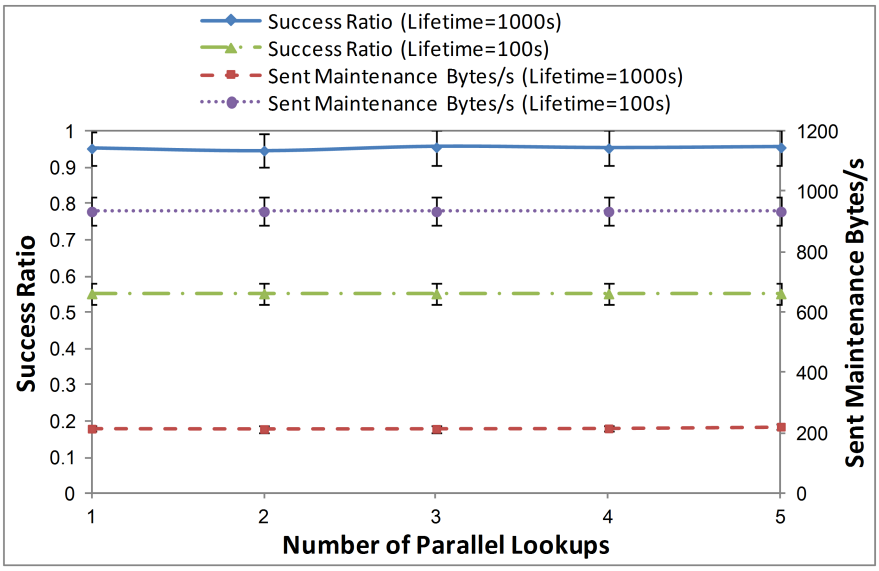



Figure 3.20: Broose: Testing on parallel lookups.

substantially affect the success rate and the maintenance traffic. Rather, the hop count (path length) to the destination node is reduced 3.24. Also, as the parallelism applies to lookup messages only, considering the lookup traffic is a better measure of cost. Figure 3.24 shows the lookup hop count and lookup traffic for varying levels of parallelism. Based on these results it has been decided to opt for a level of parallelism of 3.

## 3.5 PERFORMANCE EVALUATION

In the previous section, the most suitable parameters for Chord, Pastry, Kademlia, Broose and EpiChord have been identified considering high churn and minimum bandwidth consumption.

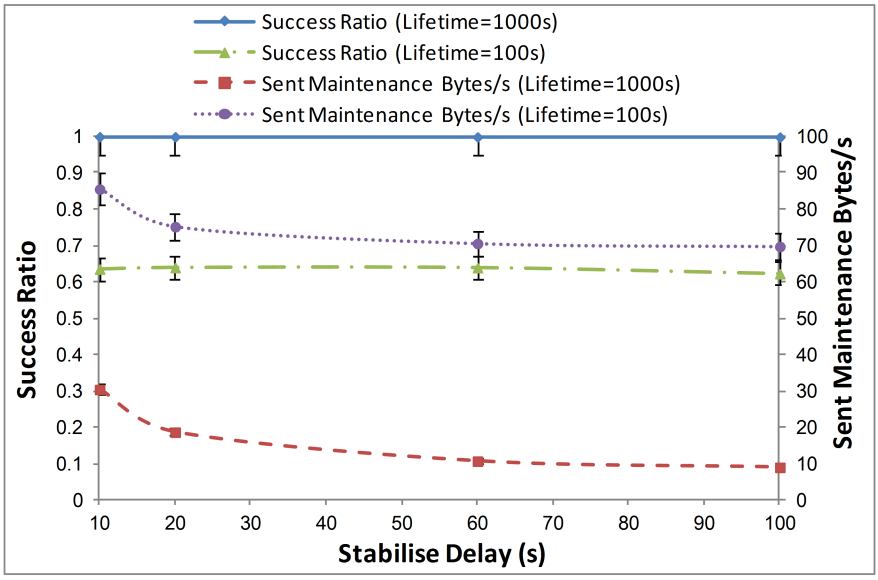



Figure 3.21: EpiChord: Testing on stabilise delay.

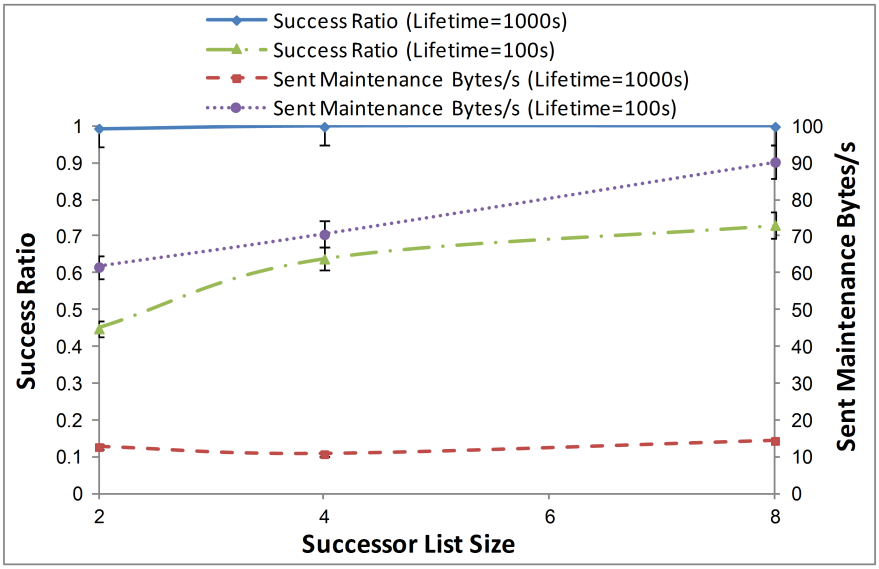



Figure 3.22: EpiChord: Testing on successor list size.

In this section, the performance of these overlays will be compared with each other using the previously selected parameters under high level of churn according to lookup success rate, hop count and bandwidth consumption.

### 3.5.1 *Effects of Churn*

In this experiment, the performance of the chosen overlays under varying levels of churn have been evaluated. It has been found that the node life time of a mobile node is varied from 150 seconds (for social networking apps) to 534 seconds (for music apps) [36]. Therefore the experimented node lifetime has been chosen from 100 seconds to 1,000 seconds. The results

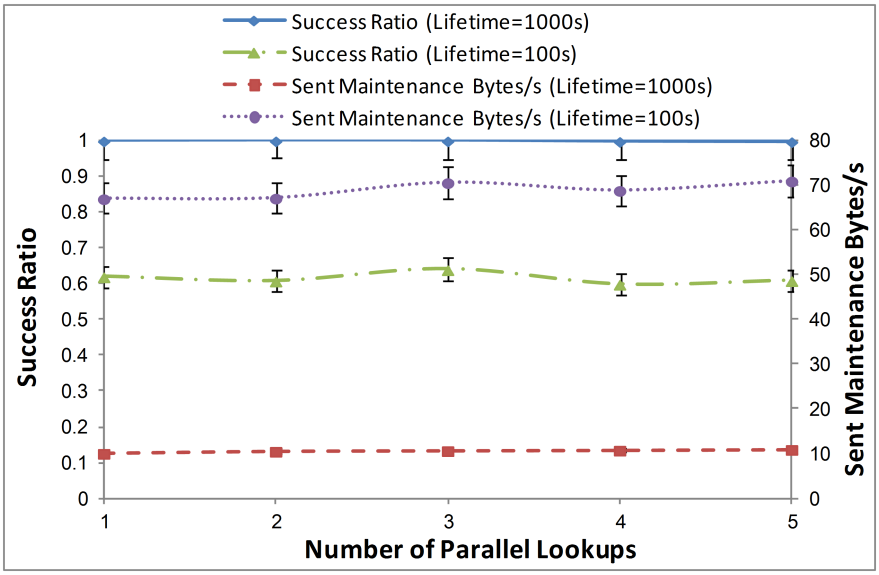



Figure 3.23: EpiChord: Testing on parallel lookups(i).

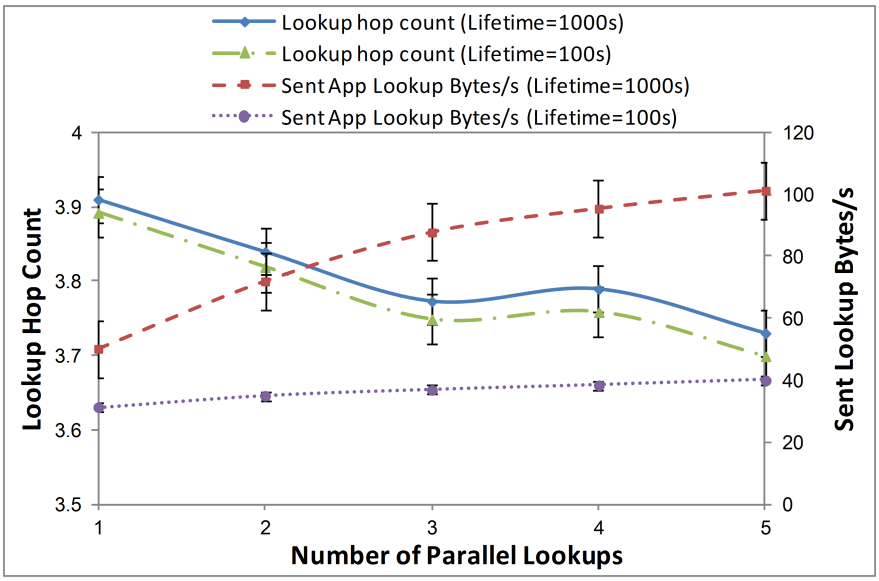



Figure 3.24: EpiChord: Testing on parallel lookups(ii).

in Figure 3.25 show the lookup success ratio under varying levels of churn from 100 seconds to 1,000 seconds. It is evident from the figure that Chord's performance is heavily affected by high levels of churn. For a node lifetime of 100 seconds the lookup success ratio collapses to a mere 2%. While this increases to about 80% as the node lifetime increases to 1,000 sec, it is clear that Chord is not a good candidate for a network consisting of mobile nodes with high churn. One likely reason for Chord's poor performance is that Chord does not immediately correct its successor and predecessor pointers of all relevant nodes as new nodes join, but waits for the periodic stabilisation process to update the pointers. Therefore, under high levels

of churn when nodes join and leave the network at a very high rate, routing table entries are not updated sufficiently frequently.

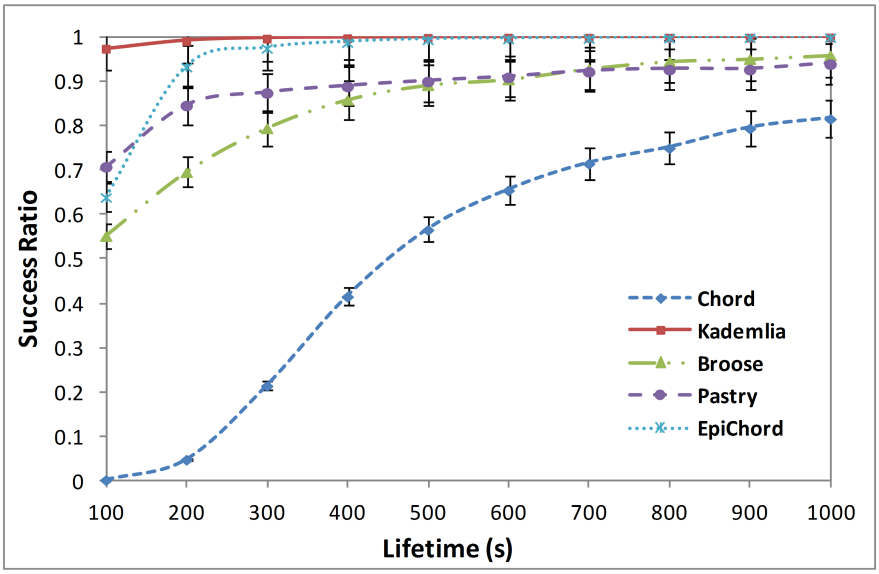



Figure 3.25: Lookup success ratio of Chord, Pastry, Kademlia, Broose and EpiChord operated on a 10,000 node networks under various levels of churn.

In Pastry, the lookup success ratio is 70% at 100 seconds of node lifetime, increasing to around 93% at 1,000 seconds of node lifetime. While this performance is acceptable, this comes at the price of the highest maintenance cost (with quite a large margin, Figure 3.26). Thus, Pastry has to be ruled out due to this cost.

Kademlia exhibits a 97% successful lookup rate in the high churn environment (100 seconds mean node lifetime). Kademlia sends parallel lookup requests to avoid timeout delays from failed nodes. Also, Kademlia updates routing table entries from data attached to lookups rather than requiring separate maintenance requests. These characteristics paired with a modest bandwidth usage and modest lookup hop count (4 for 10,000 nodes network) as shown in Figure 3.26 and 3.27 respectively, make Kademlia well suited for mobile environments.

Broose exhibits around 55% of lookup success ratio at 100 seconds of node lifetime (this increases to about 95% at 1,000 seconds of node lifetime). However, when nearly half of lookups fail in high churn networks, Broose also does not seem well suited for such churn intensive mobile environments. This is emphasised by the fact that Broose has comparatively high bandwidth costs in high churn situations (Figure 3.26).

Finally, EpiChord approached a 99% success ratio under lower levels of churn (lifetime = 1,000 seconds). In high churn, the overlay performed less favourably, with a lookup success rate of about 63%. This increases to a more usable 97% as the mean lifetime approaches 300 seconds. Like Kademlia, EpiChord uses *reactive routing* state maintenance, employing lookup response messages to carry routing table update information. EpiChord also uses parallel

lookup messages that can achieve an even better hop count performance at the cost of an increased number of lookups.

### 3.5.2 *Bandwidth Consumption*

Figure 3.26 shows the results on the required bandwidth for Chord, Pastry, Kademlia, Broose and EpiChord under varying levels of churn. In high churn environments when the node lifetime is 100 seconds, Chord and EpiChord have the lowest bandwidth cost using 26 bytes/s and 70 bytes/s respectively. However, as already discussed, Chord's lookup performance is very poor under these conditions. Pastry consumes the highest bandwidth, which is 1,674 bytes/s at this level of high churn. This is due to its expensive joining process and reactive maintenance for the routing table, leaf set and neighbour list. Broose also requires significant bandwidth at high churn levels (934 bytes/s per node at the node lifetime of 100 seconds). Kademlia consumes higher traffic (316 bytes/s at 100 seconds of node lifetime) than Chord and EpiChord, however, less than Pastry and Broose, even under high churn.

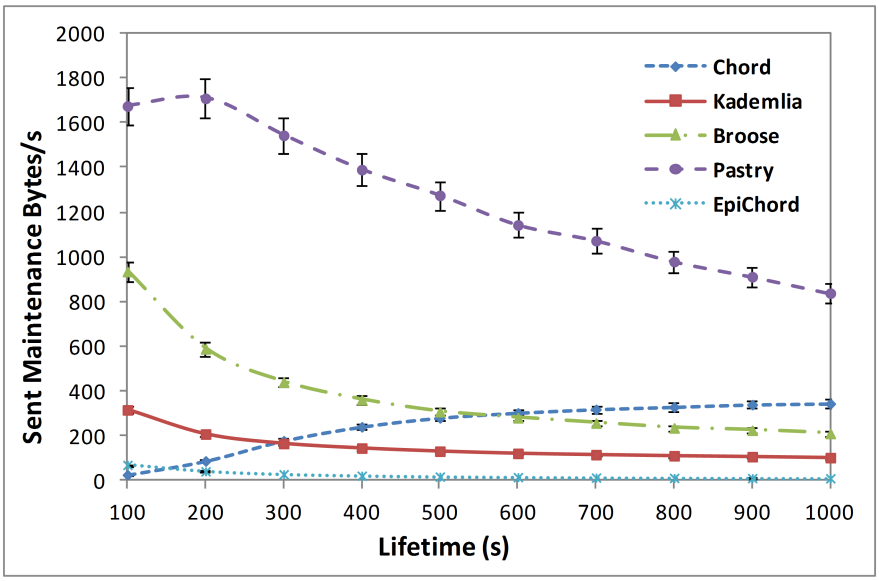



Figure 3.26: Sent maintenance bytes/s of Chord, Pastry, Kademlia, Broose and EpiChord operated on a 10,000 node networks under various levels of churn.

Unlike all other P2P overlays compared here, Chord shows an increasing amount of maintenance traffic with an increasing node lifetime. This is due to Chord periodically calling its stabilisation process and updating its finger tables (routing tables). This approach increases the bandwidth requirements as nodes stay in the network for longer. The lookup hop count for all overlays is plotted in Figure 3.27. Kademlia, Pastry and EpiChord show similar hop counts (~3.5) under varying levels of churn whereas Chord and Broose exhibit a higher lookup hop count of just over 6 and just under 6 respectively.

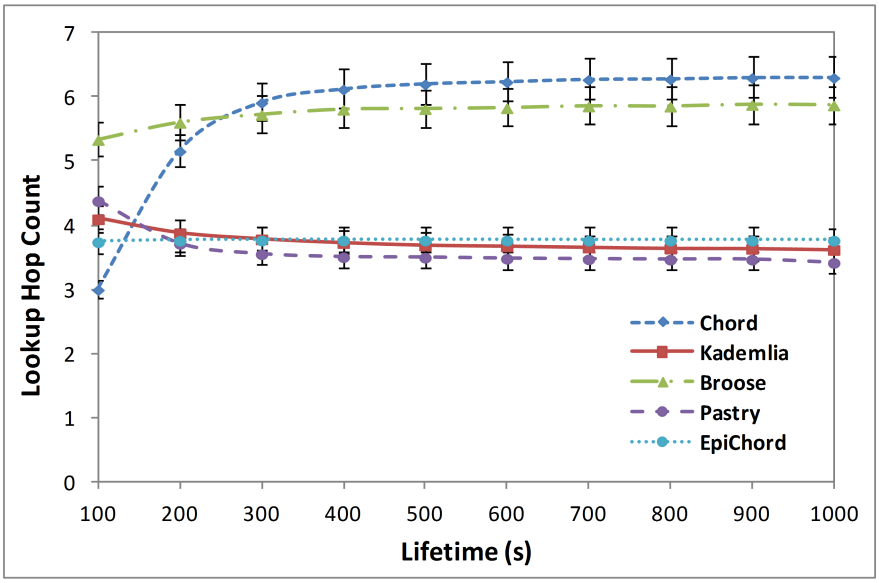



Figure 3.27: Lookup hop count of Chord, Pastry, Kademlia, Broose and EpiChord operated on a 10,000 node networks under various levels of churn.

### 3.5.3 *Effect of Different Workloads*

Leong *et al.* [47] defined two types of workloads: *Churn-Intensive* and *Lookup-Intensive*. In churn-intensive workload, each node issues on average 0.01 lookups per second, i.e. the amount of churn in the network is high relative to the lookup rate.

In the lookup-intensive workload, each node issues many more lookups during its lifetime and therefore, the amount of churn is relatively low. In this workload, each node issues 2 lookups per second on average while the rate of churn is the same as that in the churn-intensive.

The simulations that have been provided so far in this chapter featured the churn-intensive workload. However, it has been found that EpiChord's performance can be improved by introducing additional lookups to the network in a lookup-intensive workload [15]. Note that, Chord, Kademlia, Pastry and Broose do not improve significantly by introducing a lookup-intensive workload. Thus, only EpiChord will be evaluated to investigate the performance in a lookup-intensive workload and will be compared with churn-intensive workload in this section. For this particular experiment, EpiChord has been simulated with 5,000 nodes to observe its performance in different workloads under varying levels of churn from a node lifetime of 100 seconds to 1,000 seconds. The simulation parameters will be configured with the best performing parameters selected in the Section 3.4.5 for both workloads.

Figure 3.28 shows the comparison of lookup success ratio for EpiChord in lookup-intensive workload and churn-intensive workload. EpiChord exhibits almost 88% lookup success ratio in lookup-intensive workload, which is higher than in churn-intensive workload (54% lookup

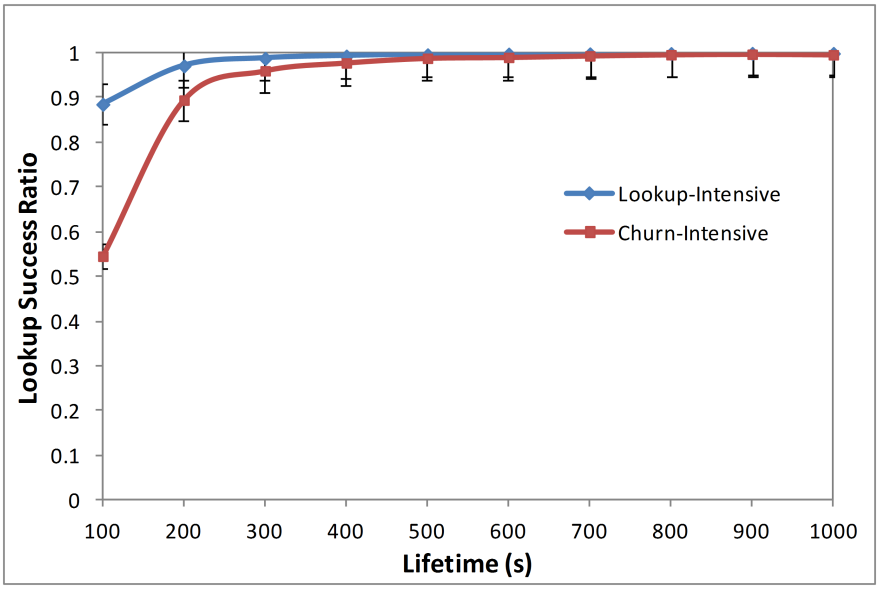



Figure 3.28: Lookup success ratio for EpiChord in lookup-intensive and churn-intensive workloads.

success ratio) in node lifetime of 100 seconds. This is due to the fact that EpiChord keeps its routing table mostly up-to-date using the large number of lookups and the associated routing table update information in the corresponding responses [47]. However, in a low churn scenario, the success ratio in both workloads are almost similar.

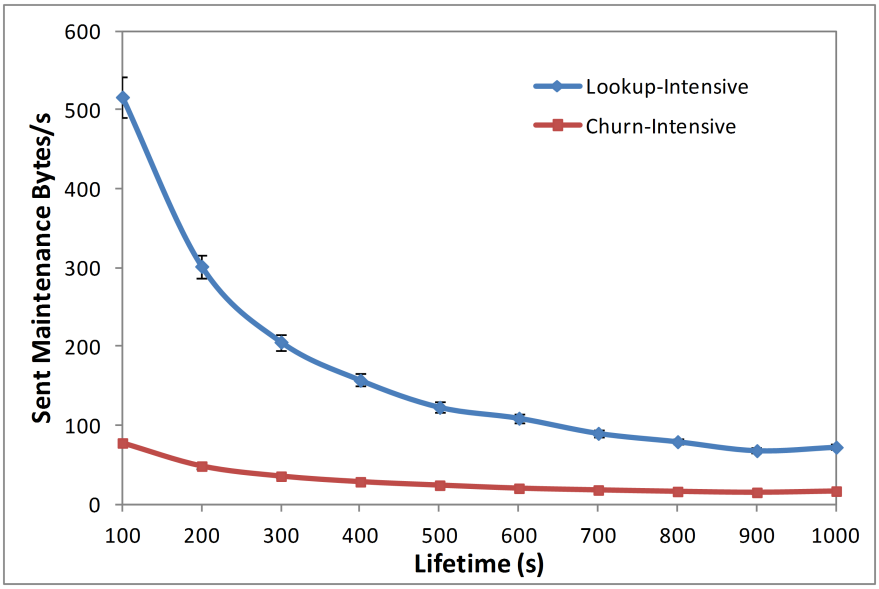



Figure 3.29: Sent maintenance bytes/s for EpiChord in lookup-intensive and churn-intensive workloads.

In the lookup-intensive scenario, each node issues a lookup request at every 0.5 seconds which is 200 times the rate of the churn-intensive workload. As a result, the lookup traffic dominates the maintenance traffic and a greater increase can be seen in lookup-intensive workload than churn-intensive workload in Figure 3.29.

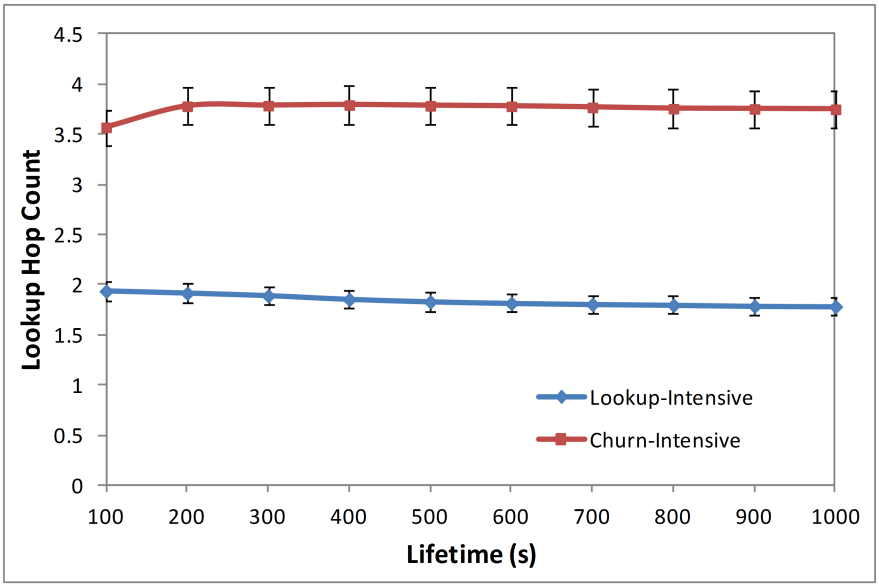



Figure 3.30: Lookup hop count for EpiChord in lookup-intensive and churn-intensive workloads.

Figure 3.30 shows the lookup hop count in EpiChord under lookup-intensive and churn-intensive workloads. In a lookup-intensive workload, EpiChord performs a successful lookup in fewer hops (on average, 1.9 to 1.7 hops in the varying lifetime from 100 seconds to 1,000 seconds) with its large routing state and therefore, performs better than churn-intensive workload (on average, 3.5 to 3.7 hops in the varying lifetime from 100 seconds to 1,000 seconds).

### 3.5.4 *Discussion*

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

- Chord's algorithm is relatively simple. However, Chord's lookup performance collapses to a mere 2% under high node churn. This is due to the inconsistency of node pointers. When a new node joins the network, Chord does not update successor and predecessor pointers of all relevant nodes immediately. It relies heavily on its periodic stabilisation process. Consequently, under high levels of churn, Chord's routing table entries become out of date and the lookup success ratio degrades drastically.

- Pastry requires considerably higher bandwidth under high churn than the other overlays. This is because Pastry nodes use parts of other nodes' routing tables to build their own routing table. Furthermore, Pastry's complex algorithm for optimising routing tables requires additional bandwidth. Pastry nodes have to repair their states by gathering information from other nodes when node failures are detected. This affects the overall

performance of success ratio and lookup hop count. Thus, it is not a good candidate for mobile environments.

- Broose exhibits an average result for lookup success but consumes more bandwidth than Chord, EpiChord and Kademlia. It also requires a higher lookup hop count than the other overlays. This is because Broose has a very exhaustive and expensive node join process. When a node joins the network, the joining node builds up its routing table using queries to other nodes for their routing table entries. Broose needs a large bucket for redundancy and cannot reduce the number of routing steps.

- Kademlia shows the best result in terms of lookup success ratio at very high churn paired with a reasonable use of bandwidth. At the same time, EpiChord shows moderate bandwidth consumption and a good lookup success performance even under high churn. EpiChord's performance can be improved further by introducing additional lookups to the network i.e. the lookup-intensive workload and even can achieve one- or two-hop lookup performance. The requirements of mobile networks where the churn rate is expected to be high and the bandwidth availability low (or expensive) matches well with Kademlia and EpiChords' high lookup success ratio, reasonable amount of bandwidth consumption and low hop count and thus, make these two candidates well suited to be used in mobile networks. Both Kademlia and EpiChord use *opportunistic maintenance* mechanisms to keep their routing tables up-to-date. In opportunistic maintenance, a node attaches routing table data to a response lookup message. The receiver then updates its routing table with this information - adding new node entries and removing node entries which are considered dead [19]. This kind of maintenance saves bandwidth as it reduces the need for dedicated messages to update routing table entries.

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

## 3.6 SUMMARY

This chapter presented a performance evaluation of Chord, Pastry, Kademlia, Broose and EpiChord in the presence of high churn and investigates their suitability for mobile networks.

Section 3.2 described the key requirements that can be used to compare against the characteristics of considering five DHT overlays to be used in mobile networks. As the mobile devices will be acting as P2P nodes, they offer lower bandwidth, therefore, a P2P overlay

should require minimal bandwidth. Again, a mobile network offers slower transmission rate than wired network, thereby the lookup delay should be as minimal as possible. Finally, a suitable DHT should handle high churn and maintain the robustness in the network.

Section 3.3 explained the experimental methodology. For the simulation, the OverSim simulation framework was used due to its flexibility with respect to underlay characteristics and possible high scalability. It is a powerful simulation tool built over OMNeT++ as the simulation engine. This section also identified the performance metrics and explained the simulation setup.

Section 3.4 evaluated the most suitable parameters for each P2P overlay. Each overlay has a number of configuration parameters that affect the performance of the overlay for different scenarios. The parameters were evaluated in high churn environments and the best performing configuration selected on the basis of maximum lookup success ratio and minimum bandwidth consumption.

After tuning the best parameters, each overlay was compared with each other to identify the most suitable overlay under churn. Section 3.5 provided a thorough analysis of performance evaluation of Chord, Pastry, Kademlia, Broose and EpiChord. The simulation results suggested that Kademlia is the most appropriate P2P overlay to implement on the mobile network according to lookup success ratio under high levels of churn. At the same time Kademlia consumes a moderate level of bandwidth (316 bytes/s) which is also acceptable in mobile networks, where the typical transfer rate is $10 - 100$ KB/s. Similarly, EpiChord also achieves a high success ratio while consuming the least amount of bandwidth (70 bytes/s in high churn) making it also a strong candidate algorithm for mobile environments. Utilising the opportunistic maintenance mechanism and lookup parallelism make Kademlia and EpiChord more efficient than the other overlays.

As stated before, this research focuses on a churn intensive networks such as the mobile networks in which Network Address Translation (NAT) plays a crucial role. For this reason, in the next chapter, the performance of different overlays under NAT will be evaluated and compared with the performance of non-NATed networks. Furthermore, NAT has different impacts on different routing approaches [10]. Hence, the performance of using different routing mechanisms will be investigated as well.

# 4

# PERFORMANCE EVALUATION OF STRUCTURED P2P OVERLAYS UNDER NAT

*The previous chapter analysed the performance of a number of common* P2P *overlays and determined the best configuration parameters to achieve optimal performance under churn. This chapter now evaluates the impact of Network Address Translation (NAT) in the performance of Chord, Kademlia and EpiChord* P2P *overlays using Iterative routing. It also compares the impact of both routing methods: Iterative and Recursive under NATed environments.*

## 4.1 INTRODUCTION

Network Address Translation (NAT) disrupts the original model of P2P connectivity as it prevents incoming connections from reaching their destination host. Enabling peers to avoid NATs requires some networking techniques, referred to as NAT traversal. However, most analytical models of overlay networks assume the underlying network to be simplistic in nature and there is no mechanism to simulate NAT in evaluation environments such as PlanetLab and OverSim (or in any other P2P simulation environments). In this chapter, a network topology is introduced where a fraction of the nodes are under NATs and a NAT traversal approach, UDP hole punching, is enabled to make sure that the NATed peers can communicate with other NATed and non-NATed peers in OverSim.

Once the NATed environment is set, it is necessary to validate the performance of a number of P2P overlays in the NATed environment, against the performance of the same P2P overlays in the non-NATed environment. This will help to compare and contrast the simulation parameters that are required to achieve the same level of performance in both environments (NATed and non-NATed) and investigate their underlying reasons.

Using iterative routing, Chord, Kademlia and EpiChord have been investigated in the NATed environment. The simulation has been done in OverSim and evaluated the performance of these P2P overlays considering the peers under high churn as well.

The rest of this chapter is organised as follows. Section 4.2 describes the basic network topology that has been used to implement NAT in OverSim and outlines the message flow of two scenarios: Peers under a common NAT and different NATs. Section 4.3 presents the performance of Chord, EpiChord and Kademlia under NATs and compares this with the performance of the same overlay without NATs. As the routing approach has a significant impact on NATed peers, the performance has also been compared with iterative and recursive

routing in Section 4.4. Section 4.5 presents the mechanism to deploy NAT and the NAT traversal technique, UDP Hole Punching without a dedicated rendezvous server, in OverSim. A brief summary of this chapter is presented in Section 4.6

## 4.2 NAT TRAVERSAL USING UDP HOLE PUNCHING WITHOUT DEDICATED RENDEZVOUS SERVERS

As mentioned in Chapter 2, the UDP hole punching technique relies on a rendezvous server to initiate and maintain the NAT traversal for peers in private networks (behind NAT). For this, it is required to ensure the continuous uptime of a pre-defined rendezvous server. This is against the general notion of P2P systems where there is no concept of a server. Therefore, an alternative method of UDP hole punching for P2P systems is proposed here that does not utilise a predefined rendezvous server.

Instead, a few randomly chosen public peers are equipped with the required functionalities so that they can emulate themselves as rendezvous servers. The public peers are assumed to have considerably higher lifetime than their private counterparts and are easily reachable from any public or private peer alike. Even so, to safeguard against the sudden unavailability of any chosen emulating rendezvous server, a resilience strategy needs to be maintained. In short, the proposed mechanism is as follows. For each private network, a couple of existing public peers are randomly chosen as its corresponding rendezvous servers. Then, each newly joined private peer in that private network communicates with one of the chosen (emulating) public peers to initiate the hole punching procedure. Two peers are chosen to ensure redundancy as part of the resilience strategy.

The proposed approach, along with the resilience strategy is explained in detail in the following subsections, considering two scenarios: peers behind the same NAT and peers behind two different NATs.

### 4.2.1 *Peers behind the Same NAT*

The first scenario illustrates the situation where peers in an overlay are behind the same NAT, however, they don't know whether the other peers are behind the same NAT. An exemplary topology of this scenario is illustrated in Figure 4.1. According to this topology, private peers A and B are the two peers who are behind the same NAT and the public peer P has been selected to act as the rendezvous server for all peers behind this NAT. There might be other public peers or even other private peers inside the same NAT. They have been excluded from the figure and the discussion for brevity. The NAT router is responsible for network address translation using a table called its NAT table. For emulating the functionalities of a rendezvous

server, Peer P utilises a table called a *Registration table* to store the public (IP) address, public port, private address, private port and the corresponding key of any private peer.

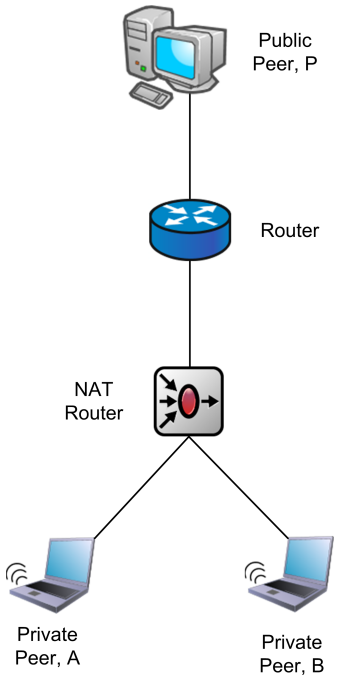



Figure 4.1: Network topology depicting two private peers under the same NAT.

Suppose Peer A and Peer B would like to communicate with each other. The message flow to establish the communication between them using the proposed mechanism is illustrated in Figure 4.2 and discussed below.

1. Peer A sends a registration message consisting of its private address-port pair and overlay key to Peer P.

2. Peer B sends a registration message consisting of its private address-port pair and overlay key to Peer P.

3. The registration message from A reaches the NAT router. The NAT router checks if there is any entry for Peer A in its NAT table, depicting the scenario that A has already interacted with P.

4. Assuming this is the first interaction between A and P, the NAT router retrieves the source address (the private address of A) and port number from the registration packet. It then generates an unused port number for A and creates a new entry in its NAT table containing the private address and port of A and its own public IP address and the newly generated port number. Finally, it replaces the source address and port number of the registration packet with its public address and the newly generated port number and forwards the packet to P.

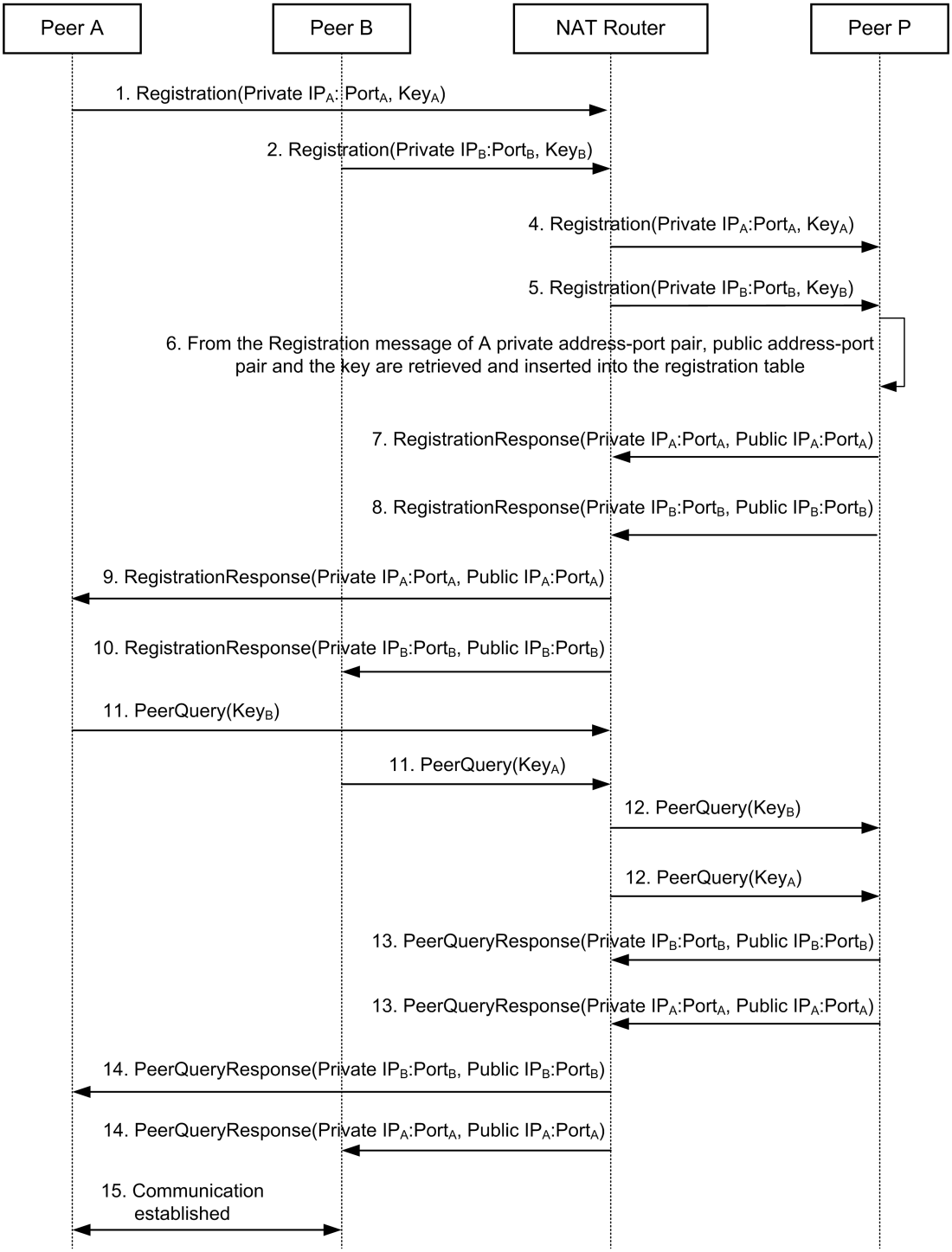



Figure 4.2: Message flow for establishing communication between private peers under the same NAT

5. Similarly, the registration message from B reaches the NAT router and the NAT router forwards the packet to P after creating a respective entry in its NAT table.

6. Upon receiving the registration message from A, P extracts the private address, private port and the key from the message. In addition, it retrieves the public address and port number from the packet. Then, it checks in its *Registration table* if there is an entry in the

table containing all this information. Assuming no such entry exists, P creates an entry with this information in the registration table.

7. Then, P replies back with a registration response - containing the private address, private port, public address and public port retrieved from the registration packet in the previous step - which is destined for the same public address and port.

8. Similarly, upon receiving the registration message from B, P extracts the required information, creates an entry in its registration table and replies back with the registration response containing the private address, private port, public address and public port retrieved from the registration packet. The response is sent back to the NAT router.

9. The registration response for A reaches the NAT router which initiates a reverse network translation process by extracting the destination address and port of the packet. Then, it retrieves the corresponding private address and port from its NAT table by using the extracted public address and port. Finally, it forwards the response packet to A.

10. Similarly, upon receiving the registration response for B, P initiates the reverse network translation and finally, forwards the response to B.

11. Next, A and B would like to communicate with each other. Both of them send query packets (depicted as *PeerQuery* in Figure 4.2) containing the key of the other peer.

12. The NAT router forwards both query packets to P using the same approach as described before.

13. P replies back to A and B with query responses containing the private address, private port, public address and public port of the queried peer which ultimately reach the NAT router.

14. The NAT router, like before, forwards the respective packets to A and B.

15. Upon receiving the query response packets, A and B learn the private address-port and the public address-port pairs of the other peer. Since both of them reside in the same NAT, they can either use the private address-port pair or the public address-port pair of the other peer to communicate with each other subsequently via the NAT router, indicating an established communication between A and B.

### 4.2.2 *Peers behind Different NATs*

This scenario depicts the situation when two peers residing in different NATs would like to communicate with each other. An exemplary topology of this scenario is illustrated in Figure 4.3. According to this topology, private peers A and C are the two peers that would

like to communicate with each other. They belong to two different NATs, *Private Network* 1 and *Private Network* 2 respectively, and are under NAT routers M and N respectively. The public peer P1 has been selected to act as the rendezvous server for all peers under NAT router M, whereas the public peer P2 has been selected to act as the rendezvous server for all peers under NAT router N. In addition, both M and N maintain another table called *Incoming*; enlisting which peers (having a particular address and port) are allowed to contact a peer behind the respective NAT.

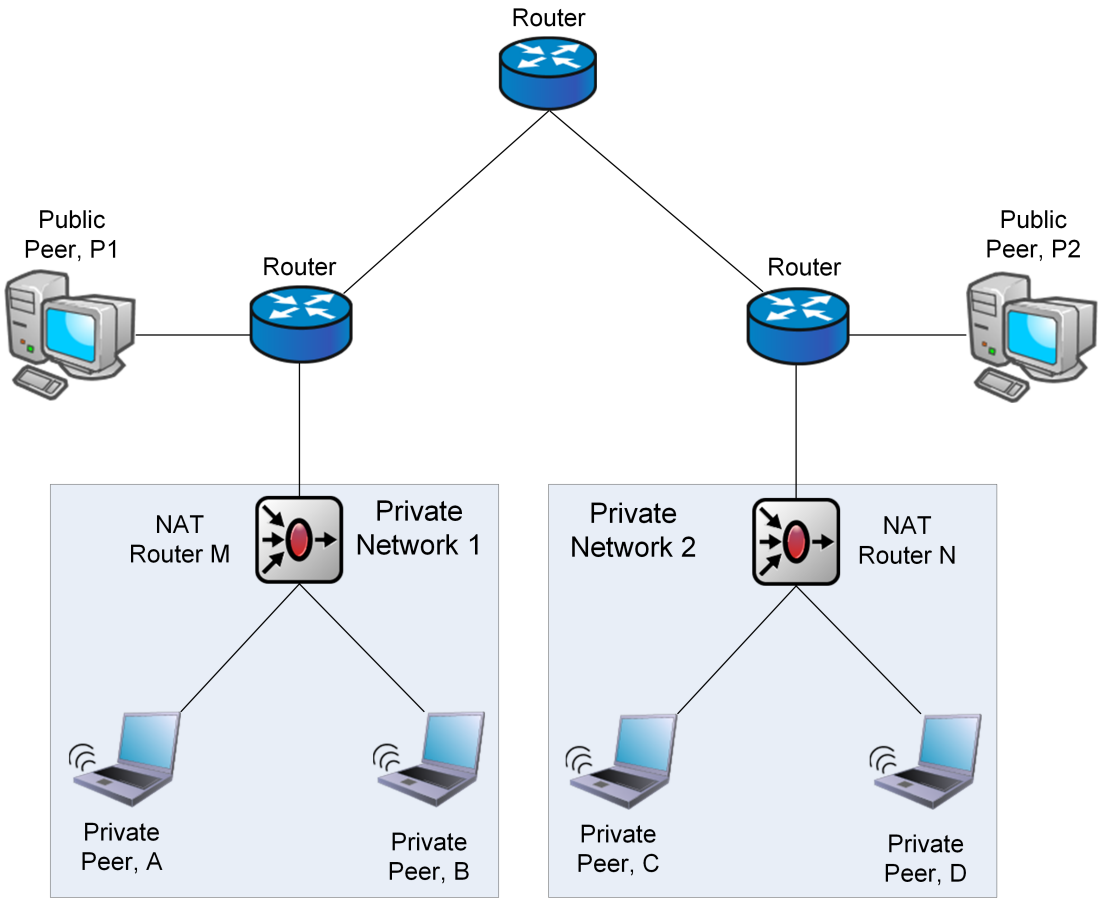



Figure 4.3: Network topology depicting two private peers residing in different NATs.

Suppose peer A and peer C need to communicate with each other. The message flow to establish the communication between them using the proposed mechanism is illustrated in Figure 4.4. The message flow is similar to the one in Section 4.2.1 and is outlined below. For brevity, it is assumed that Peer A has registered with Peer P1 whereas Peer C has registered with Peer P2 using the mechanism described previously and both peers have received the registration responses back from P1 and P2 respectively. The rest of the message flow is discussed below.

1. A sends a query request for C to P1 via NAT router M. Similarly, C sends a query request for A to P2 via NAT router N.

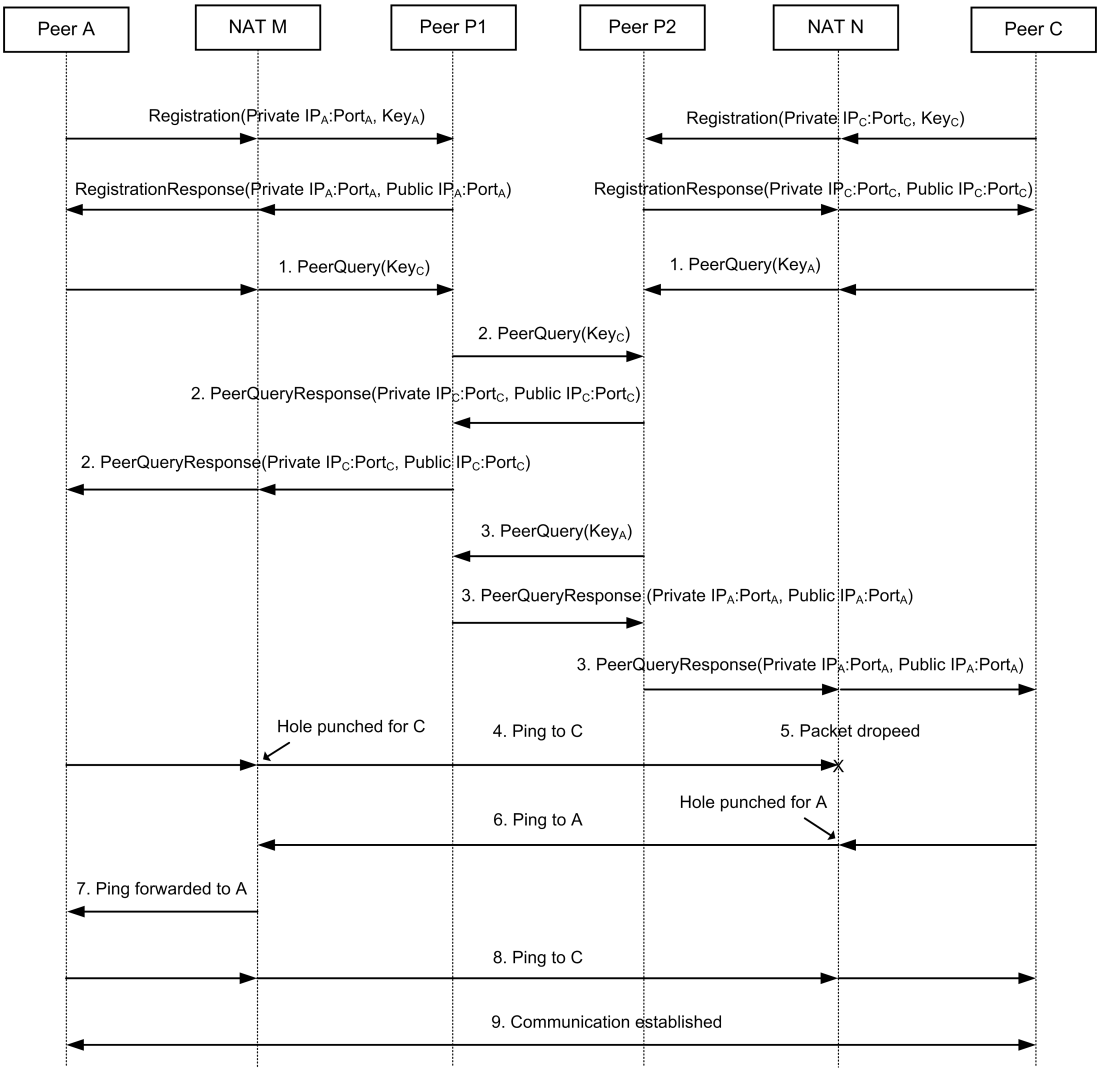



Figure 4.4: Message flow for establishing communication between private peers in different NATs

2. Since P1 does not have any information regarding C, it forwards the query request to P2. P2 returns a query response containing the private address, private port, public address and public port of C back to P1 which is then returned back to A. From this response, A learns the public address and public port of C.

3. Similarly, P2 retrieves the query response from P1 containing the private address, private port, public address and public port of A which is then returned back to C. Ultimately, C learns the public address and public port of A.

4. A sends a PING packet to C using the public address and port of C via M. When M receives this packet, it creates an entry for A with the destination address and port of (C) this packet in its *Incoming* table. This essentially punches a hole for C in M so that C can interact with A. The packet is then forwarded by M and finally reaches N.

5. N checks in its *Incoming* table to determine if there is an entry for A with respect to C. Assuming there is no such entry, the packet is dropped.

6. Next, C sends a PING packet to A using the public address and port of A via N. When N receives this packet, it creates an entry for A with the destination address and port of (A) this packet in its *Incoming* table. This essentially punches a hole for A in N so that A can interact with C. The packet is then forwarded by N and finally reaches M.

7. M checks in its *Incoming* table to determine if there is an entry for C with respect to A. As an entry is found indicating that the communication from C is allowed by A, the PING packet is forwarded to A.

8. A again sends a PING packet to C via M. The packet finally reaches N. Since an entry is found indicating that the communication from A is allowed by C, the PING packet is forwarded to C.

9. Now the hole punching procedures for both A and C are complete and both peers can communicate with each other subsequently via the respective NAT routers, indicating an established communication between A and C.

### 4.2.3 *Resilience Strategy*

With the absence of a dedicated rendezvous server, the proposed approach emulates the functionalities of a rendezvous server using existing public peers. Since the emulated rendezvous server holds the registration information for the respective private peers, this information will be lost if the emulated server suddenly leaves the network. If this happens, all the corresponding private peers will be required to re-register again with another chosen emulated rendezvous server. As a resilience strategy to guard against this possibility, two public peers are randomly chosen as rendezvous servers for each private network. All peers in a private network may utilise any of the respective rendezvous servers for the message flow. The rendezvous servers communicate with each other at predefined intervals to exchange their corresponding registration tables to ensure consistency and redundancy. When one rendezvous server leaves the network, another public peer is randomly chosen to replace its place and the existing rendezvous server communicates with the new rendezvous server to exchange the registration table.

Utilising more than one emulated rendezvous server has additional advantages. The communication burden of a single emulated server for a private network containing hundreds of peers can be quite exhaustive. This burden is halved and shared when two rendezvous servers are selected. It can be argued that this burden can further be reduced by selecting more than two public peers as rendezvous servers which may also reduce the probability of all rendezvous servers leaving the network simultaneously. However, it must be considered

that selecting a large number of rendezvous servers will substantially increase the network maintenance traffic.

Also, it must be noted that this resilience strategy will fall apart if both rendezvous servers leave the network almost simultaneously. An additional mechanism will be needed to ensure that this does not happen. One such mechanism in the simulated environment of Oversim will be discussed in Section 4.5.2.

## 4.3 PERFORMANCE ANALYSIS IN NATED AND NON-NATED NETWORKS

This section validates the performance of the chosen P2P overlays: EpiChord and Kademlia, suitable for mobile networks in NATed environments against the performance of the same P2P overlays in non-NATed environments. Chord has also been evaluated in this section as it supports both iterative and recursive routing and different routing approaches have different impacts over NATs [10].

### 4.3.1 *Network Setup*

- ***NATed Environment:*** The simulation has been set up under two private, i.e. NATed networks where the total number of nodes are varied between 1,000 and 5,000. Among the total number of nodes, 20% are public nodes (with public IP addresses) and the remaining 80% are private nodes equally divided between two private networks.
- ***Non-NATed Environment:*** The simulation has been set up under a public network where the number of nodes range from 1,000 to 5,000 nodes, all with public IP addresses.

### 4.3.2 *Simulation Setup*

- ***Churn:*** Two levels of churn have been experimented with. When the node lifetime is set as 100 seconds, it is considered to be high churn, whereas for a node lifetime of 1,000 seconds, it is considered to be medium churn.
- ***Lookup Interval:*** Each node issues lookups for random keys at intervals exponentially distributed with a mean of 60 seconds.
- ***Transport Protocol:*** UDP has been used as the transport protocol in the simulation.
- ***Routing:*** All the overlays have been evaluated using iterative routing under churn as in Chapter 3. In this section, the overlays will be simulated again using **Iterative** routing. However, different routing techniques have different impacts on NATs. Section 4.4 will compare the performance of the overlays using both routing algorithms: Iterative and Recursive, in the NATed environment.

- ***Repetition:*** Similar to the previous experimentation, each configuration has been repeated 5 times, and results have been averaged. In the plots, the 95% confidence intervals have been calculated across the repetitions, but they are sometimes too small to be visible.

### 4.3.3 *Chord*

A number of key parameters of Chord have been evaluated to achieve the best possible performance under churn in Chapter 3. The optimised parameters along with the values are listed in Table 4.1 and this configuration will be used throughout this chapter.

Table 4.1: Simulation parameters for Chord.

| **Parameters** | **Value** |
|---|---|
| Stabilise Delay (s) | 20 |
| Fixfinger Delay (s) | 30 |
| Successor List Size (nodes) | 8 |
| Check Predecessor Delay (s) | 5 |
| Size of Extended Finger Table | 0 |

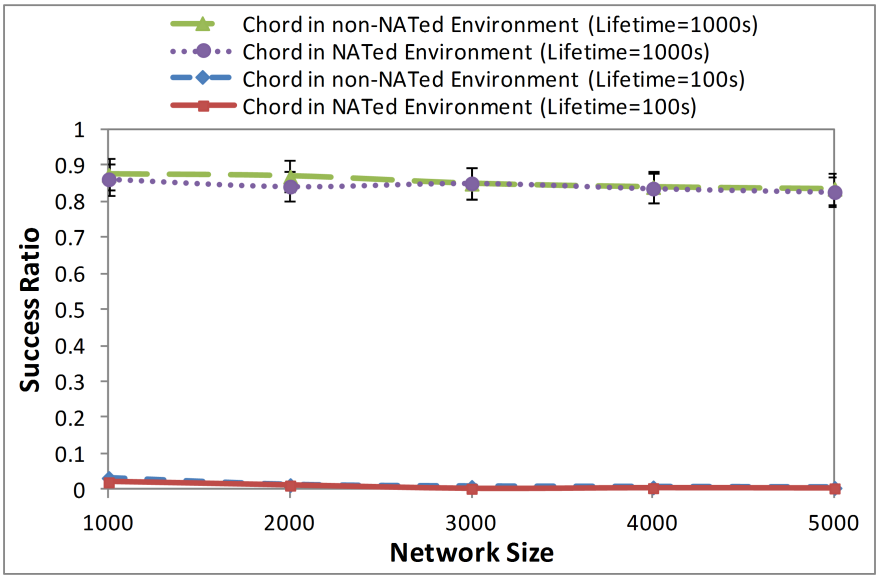



Figure 4.5: Chord: Lookup success ratio under NATed and non-NATed environments.

Figure 4.5 to 4.7 present the results of success ratio, bandwidth consumption and hop count for Chord under NATed and non-NATed environments.

Comparing the lookup success ratio under NATed and non-NATed environments, Figure 4.5 shows that the success rate reduces with network size in both cases. Also the success rate

for both churn scenarios matches considerably with each other in both environments. For a medium level of churn (node lifetime of 1,000 seconds); Chord performs quite well, whereas its performance degrades drastically in a high churn level (node lifetime of 100 seconds). This is due to the routing table not being updated efficiently.

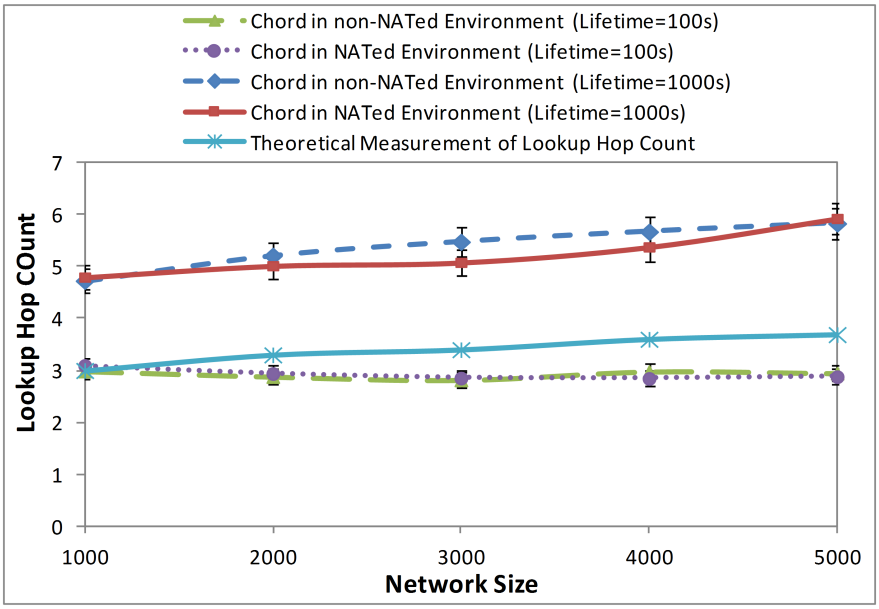



Figure 4.6: Chord: Lookup hop count under NATed and non-NATed environments.

Chord nodes require $O(\log N)$ hops for a lookup query, where N is the number of nodes [74]. Figure 4.6 presents the lookup hop count of Chord over the network size where the result quite closely matches the theoretical measurement for the medium level of churn. But under high churn, the lookup hop count has been found to be higher than what is found in medium churn. Again the reason is that Chord could not update the successors and predecessors in its routing table as it has to wait for the stabilisation process, and if the nodes join and leave at a very high rate, lots of stale entries remain in the table. But the results plotted in the figure shows that the mean lookup hop count is also a close match in both NATed and non-NATed environments as well as in both churn scenarios.

Figure 4.7 shows the effect of bandwidth consumption of the Chord overlay in NATed and non-NATed environments. The sent maintenance bytes per second per node has been considered as bandwidth consumption (as mentioned in Chapter 3). Clearly in the NATed environment, Chord utilises a slightly higher amount of maintenance bytes than in the non-NATed environment. This is because of the following reasons:

- The UDP hole punching technique has been used to connect peers behind NATs which uses extra bytes to send and receive requests and responses to register the private peers with public peers (emulated rendezvous servers).

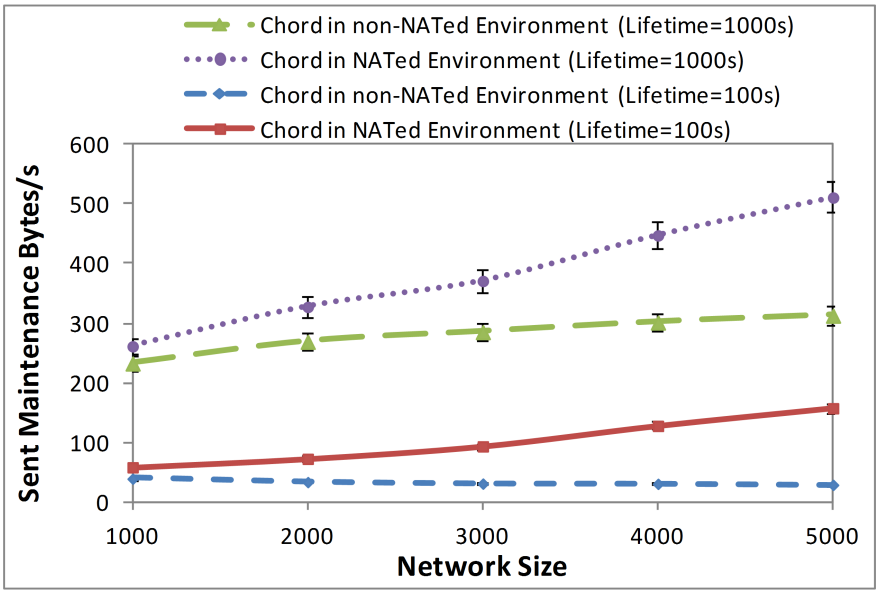



Figure 4.7: Chord: Sent maintenance bytes/s under NATed and non-NATed environments.

- To exchange the entries of the registration table between emulated rendezvous servers and for sending queries regarding other nodes.
- To punch new connections (or holes) in different NAT routers.

In high churn scenarios, i.e. when the lifetime of a node is 100 seconds, the number of sent maintenance bytes are less than the medium churn scenario (when the lifetime of a node is 1,000 seconds). This is because in high churn scenarios, there are a very few successful requests and therefore, there are fewer messages in the network. When the node lifetime increases, the Chord nodes become stable. In such cases, the nodes issue an increasing number of stabilisation call and update the fixfinger table which attribute to the increase of maintenance bytes. When the network size grows, the amount of traffic also increases linearly and NATed nodes consume a comparatively higher amount of bandwidth than the public nodes.

These results show that similar performance can be achieved in both NATed and non-NATed environments. However, Chord is not a good candidate to be used in churn intensive NATed networks such as in mobile networks because of its poor lookup performance in high churn.

### 4.3.4 *EpiChord*

EpiChord has been selected as one of the best overlays to be used in churn intensive mobile networks, as shown in Chapter 3. This section will evaluate the performance under the NATed environment.

The most suitable parameters that were selected for churn intensive mobile networks under churn in Chapter 3 are listed in Table 4.2 and will be used throughout this chapter.

Table 4.2: Simulation parameters for EpiChord.

| Parameters | Value |
|---|---|
| Stabilise Delay (s) | 60 |
| Successor List Size (nodes) | 4 |
| No. of Parallel Lookups | 3 |

Figure 4.8 to 4.10 present the results of success ratio, hop count and bandwidth consumption respectively for EpiChord under NATed and non-NATed environments.

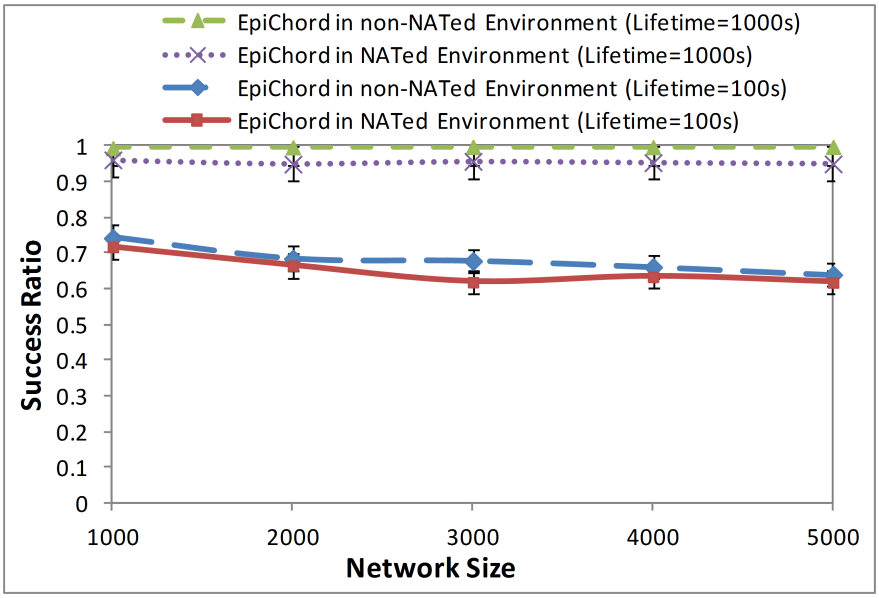



Figure 4.8: EpiChord: Lookup success ratio under NATed and non-NATed environments.

In Figure 4.8, it is clear that in both environments, EpiChord shows a very similar success ratio. Once the NATed peers register with the public peer (emulated rendezvous server) and punch holes with respect to each other, they can send lookup messages exactly in the same way as if they were public peers. In medium levels of churn, EpiChord exhibits an almost 99% success ratio in the non-NATed environment and a 96% success ratio in the NATed environment. By using reactive routing state maintenance, EpiChord can update its routing table frequently. EpiChord also uses parallel lookup messages that guarantees EpiChord to find a lookup successfully. In the high churn scenario, EpiChord generates a 74% to 64% success ratio in non-NATed networks, and a 71% to 62% success ratio in NATed networks over network sizes from 1,000 to 5,000 nodes. Compared to Chord, EpiChord shows much more efficiency with regard to lookup success ratio under high churn.

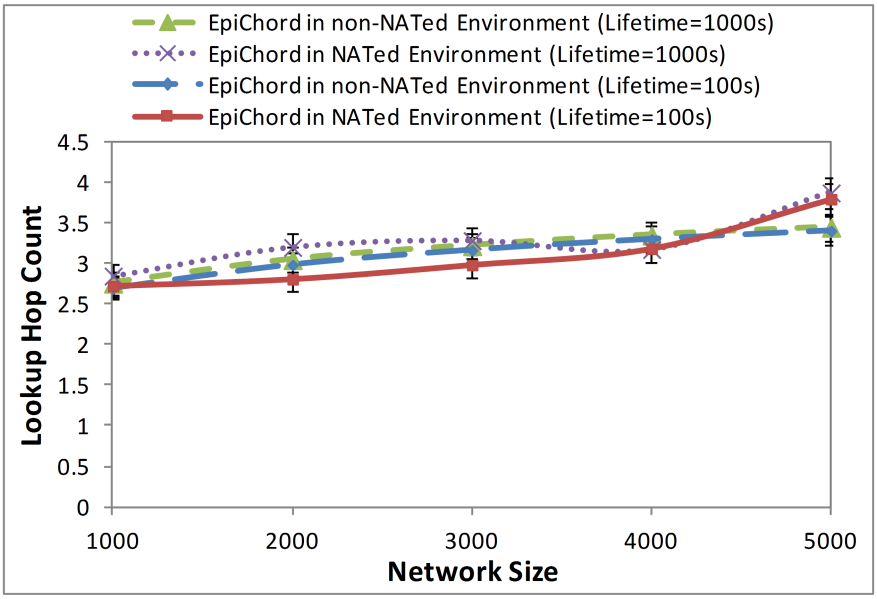



Figure 4.9: EpiChord: Lookup hop count under NATed and non-NATed environments.

Similarly, Figure 4.9 exhibits a closely matched lookup hop count in both NATed and non-NATed environments. This is an expected behaviour as the hop count does not depend on the underlying configuration of the network. Instead, this depends on the overlay structure which remains the same in both environments. It is also interesting to see that the lookup hop count does not differ much in medium and high churn scenarios, however, the hop count increases with the growing number of network size. In this experiment, EpiChord has been evaluated using a *Churn-Intensive* workload, i.e. each node issues lookups at an interval of 60 seconds and generates $O(\log N)$ hop performance, where N is the number of nodes. Although EpiChord can generate $O(1)$ hop performance in *lookup-intensive* workload, as described in Section 3.5.3.

Next, the effect of bandwidth consumption of EpiChord in NATed and non-NATed environments is shown in Figure 4.10. Unlike the other two performance metrics, there is a clear difference in this performance metric from NATed to non-NATed networks. This behaviour in EpiChord is caused by additional messages needed for any private peer registering with the emulated rendezvous server and to exchange the entries of the registration table between the other emulated rendezvous servers. As the network grows, the number of messages also increases in NATed environments and thus, there is an increase in the plot under the NATed environment. However, unlike Chord, EpiChord consumes more bandwidth in high churn scenarios than the medium churn scenario. When the rate of node join/leave increases, the sent maintenance bytes/s/node increases to keep the routing table fresh and current, whereas in medium levels of churn, as a node stays for longer in the network, the routing table becomes updated quickly and is large enough, therefore, the number of consumed maintenance bytes decreases.

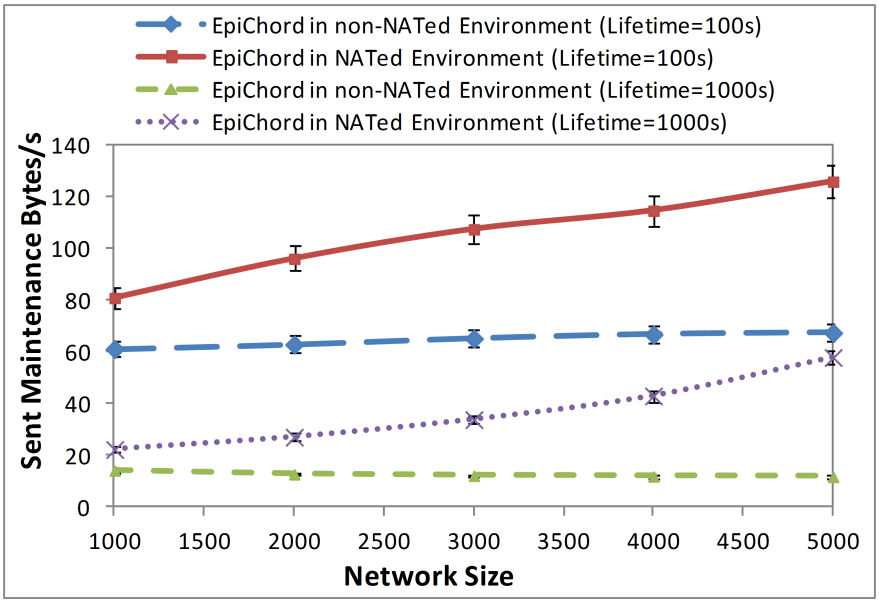



Figure 4.10: EpiChord: Sent maintenance bytes/s under NATed and non-NATed environments.

The experiment shows that similar performance can be achieved in both NATed and non-NATed environments. Although the overlay consumes extra bandwidth, the lookup success ratio has a minimal impact in the NATed environment.

### 4.3.5 *Kademlia*

In this section, Kademlia has been simulated to analyse the performance under NATed and non-NATed environments. Table 4.3 lists the parameters that have been found to be the most suitable to be used in churn intensive scenarios such as in mobile networks and these parameters will be used for all the experimentations in this chapter.

Table 4.3: Simulation parameters for Kademlia.

| **Parameters** | **Value** |
|---|---|
| Bucket Size, k | 8 |
| Siblings, s (nodes) | 2 |
| Bits per Digit, b (bits) | 1 |
| Lookup Redundant Nodes, r (nodes) | 8 |
| Bucket Refresh Interval (s) | 1,000 |
| No. of Parallel Lookups | 3 |

Figure 4.11 to 4.13 present the results of success ratio, bandwidth consumption and hop count respectively for Kademlia in NATed environments.

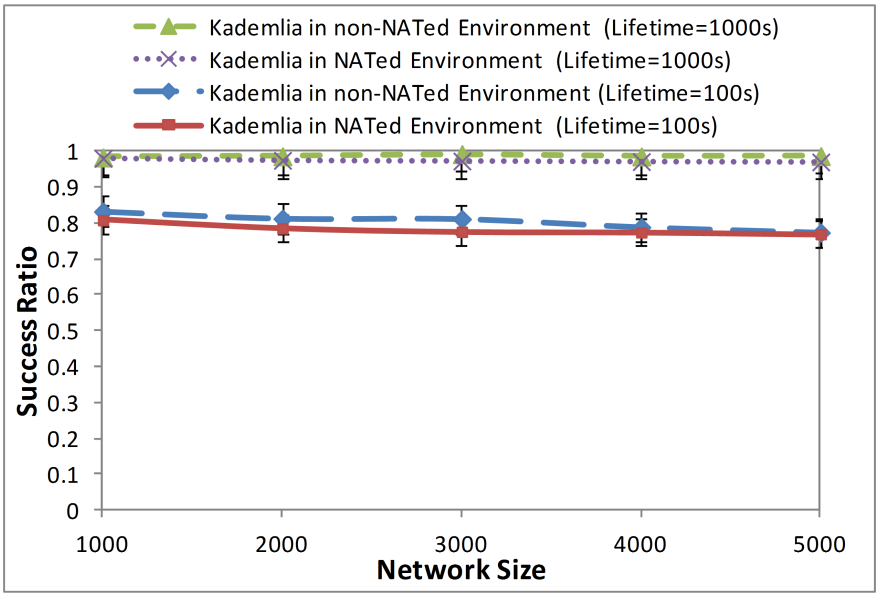



Figure 4.11: Kademlia: Lookup success ratio under NATed and non-NATed environments.

The lookup success ratio results of Kademlia have been plotted in Figure 4.11. In a high churn environment with growing network size, Kademlia exhibits around an 83% to 77% success ratio in non-NATed environments, however there is a slight decrease (around an 80% to 76% success ratio) in NATed environments. In a medium churn environment, Kademlia produces a stable success ratio (almost 98%) in both NATed and non-Nated networks. This is due to the general routing efficiency of the topology used here and how routing information is updated. Kademlia sends parallel lookup queries and exchanges the routing table entries during lookups instead of sending separate requests for maintenance. These mechanisms make Kademlia more efficient.

Figure 4.12 shows the lookup hop count results. In high churn scenarios, when the node lifetime is 100 seconds, the lookup hop count ranges from 2.8 to 3.8 and 2.9 to 3.9 over network sizes ranging from $1,000$ to $5,000$ in public networks and NATed networks respectively. In medium churn levels, such as where nodes have a lifetime of $1,000$ seconds, the lookup hop count ranges from 2.4 to 3.3 and 2.8 to 3.5 in public networks and NATed networks respectively over network sizes ranging from $1,000$ to $5,000$. There is no substantial change of lookup hop count under the two churn scenarios over network size. Therefore, it is evident that the lookup hop counts in both network environments as well as in both churn scenarios are a close match.

Figure 4.13 shows the measured bandwidth consumptions. While Kademlia has been experimented in public networks, the sent maintenance bytes varied (from $\sim$ 287 bytes/s to $\sim$ 336 bytes/s in high churn and $\sim$ 85 bytes/s to $\sim$ 122 bytes/s in medium churn) over

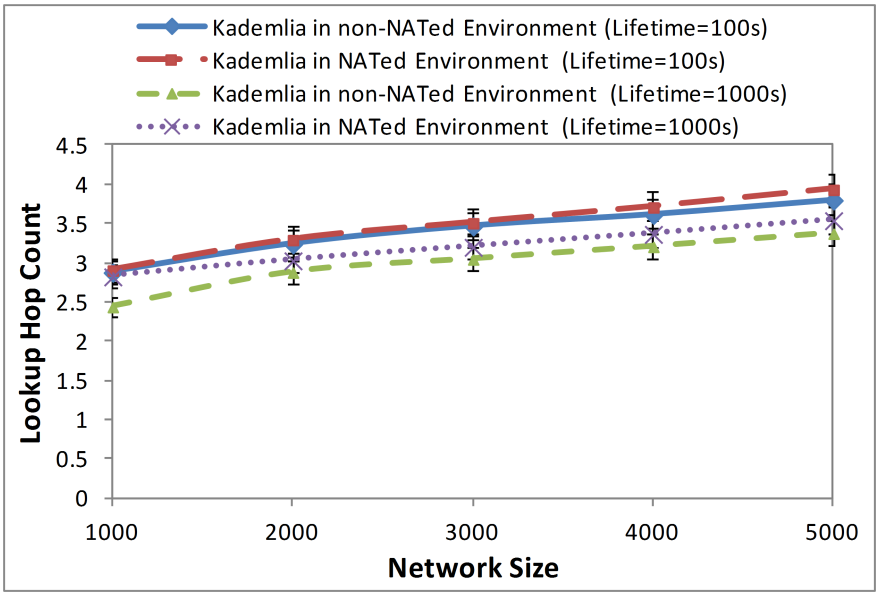



Figure 4.12: Kademlia: Lookup hop count under NATed and non-NATed environments.

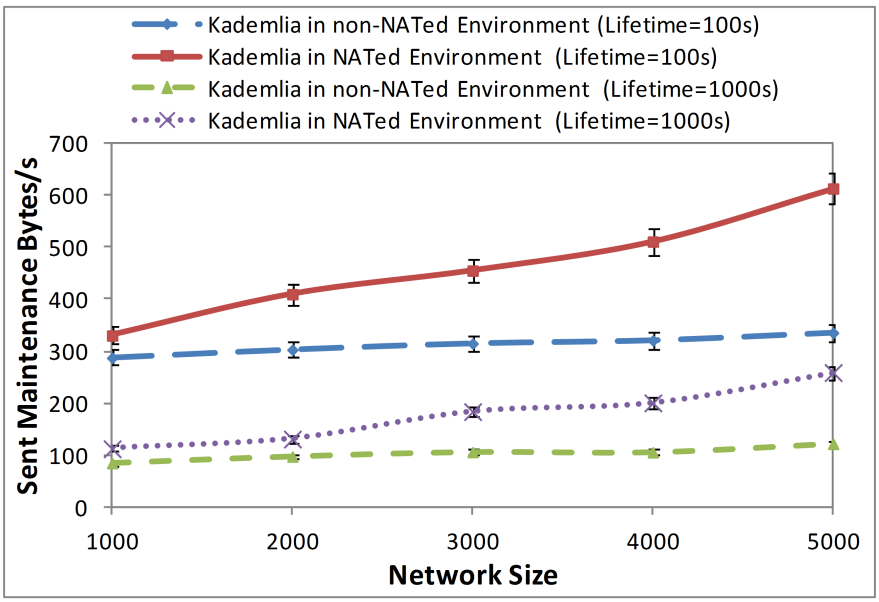



Figure 4.13: Kademlia: Sent maintenance bytes/s under NATed and non-NATed environments.

network size ranging from 1,000 to 5,000 nodes. However, the sent maintenance bytes increase substantially over network sizes in NATed networks (from ~ 330 bytes/s to ~ 613 bytes/s in high churn and ~ 113 bytes/s to ~ 259 bytes/s in medium churn). This is due to the same reason, as explained in Chord and EpiChord, where the private peers use extra bytes to traverse NATs. Again, Kademlia consumes less bandwidth in the medium churn scenario than the high churn scenario. As in medium churn, the nodes settle down and do not need to send more maintenance traffic in the network.

### 4.3.6 *Discussion*

Based on the simulation results of Chord, Kademlia and EpiChord behind NATs and within public networks, the following observations can be made:

- Chord is the most common and widely used structured P2P protocol, however, its lookup performance degrades to almost 2% under high churn scenario. Therefore, Chord is not a reliable overlay to be used in mobile networks where churn and NAT are two fundamental issues, even though Chord has been simulated to validate the results in NATed environments.
- From the previous chapter, Kademlia and EpiChord have been chosen as the best overlays to be used in mobile networks; therefore, it was necessary to investigate both these overlays under NATed environments. According to the lookup success ratio and the lookup hop count, both overlays produce similar results behind NATs and within public networks. The main difference has been found in bandwidth consumption. Both overlays, EpiChord and Kademlia, consume additional bandwidth to maintain the overlay in NATed environments. This is due to the following reasons: i) To use the UDP hole punching technique to connect peers behind NATs, the private peers use extra bytes to send and receive requests and responses to register themselves with public peers (emulated rendezvous servers), ii) To exchange the entries of the registration table between emulated rendezvous servers and for sending queries regarding other nodes and iii) To punch new connections or holes in different NAT routers. With increasing network sizes, it increases as well. Also Kademlia consumes a lot higher bandwidth than EpiChord. For example, in NATed environments, Kademlia consumes $\sim 330$ bytes in high churn and $\sim 113$ bytes/s in medium churn with a network size of $1,000$ nodes (Figure 4.13); whereas EpiChord consumes $\sim 80$ bytes/s in high churn and $\sim 20$ bytes/s in medium churn with the same network size (Figure 4.10).
- It should be noted that EpiChord's performance can be improved further using a *Lookup-Intensive* workload as shown in Section 3.5.3 in NATed environments. However, iterative routing requires additional $i-1$ NAT holes to be punched, where $i$ is the total number of querying nodes in the network and using this approach in EpiChord over a NATed environment opens a potential security breach in the network.

## 4.4 Performance comparison with iterative and recursive routing under NAT

In Section 2.5.2 of Chapter 2, it has already been discussed that routing technique has an impact over NAT. But until now, all the experiments have used iterative routing. As iterative

routing works better in high churn environments [76], the overlays have been simulated using iterative routing. In this section, the performance of Chord and Kademlia will be evaluated using both routing approaches: iterative and recursive routing in NATed environments. EpiChord does not support recursive routing; therefore, EpiChord has not been used in these experiments.

The NATed environment will be simulated as described in Section 4.3.1. The simulation setup will also be the same as described in Section 4.3.2. The only difference will be here with the routing mechanism. In this section both iterative and recursive routing will be simulated.

### 4.4.1 *Chord*

Chord has been analysed using iterative and full-recursive routing. The simulation parameters are the same as listed in Table 4.1 in Section 4.3.3. The comparison of the results are shown in Figures 4.14 and 4.15. To observe the effect of churn, the simulation also considers two levels of churn by varying the node lifetime mean between 100 seconds for high churn and 1,000 seconds for medium churn.

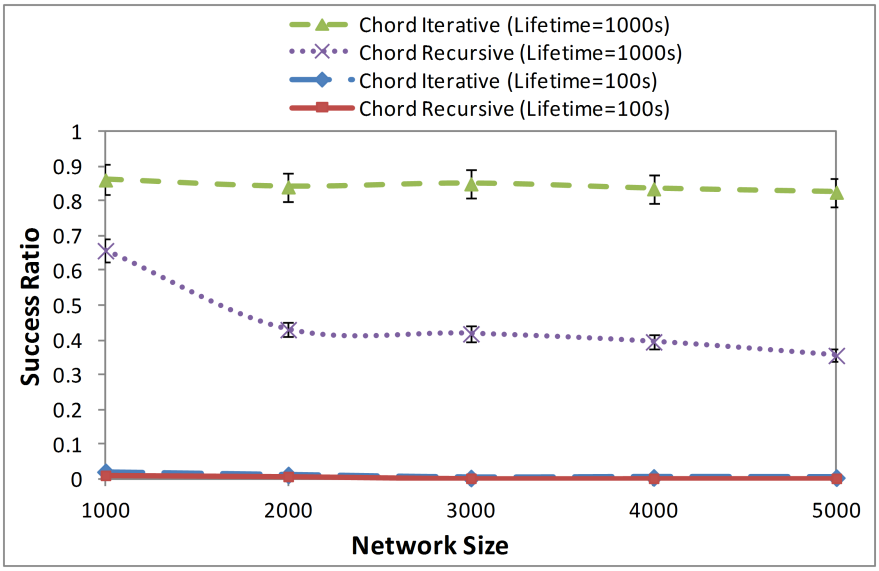



Figure 4.14: Chord: Lookup success ratio in iterative and recursive routing under NATed environments.

The lookup success ratio of two routing schemes of Chord has been shown in Figure 4.14. From the figure, it is clear that the success ratio is higher in iterative routing in a medium churn scenario (when the lifetime is 1,000 seconds), whereas the recursive routing shows a lower success ratio. In iterative routing, the lookup success ratio is almost 86% to 82% with the increasing network sizes from 1,000 nodes to 5,000 nodes compared to the recursive routing where the lookup success ratio varied from 65% to 35%. In a high churn environment (when the lifetime is 100 seconds), the performance of success ratio in both routing approaches degrades drastically. This is an expected behaviour as described in Section 4.3.3.

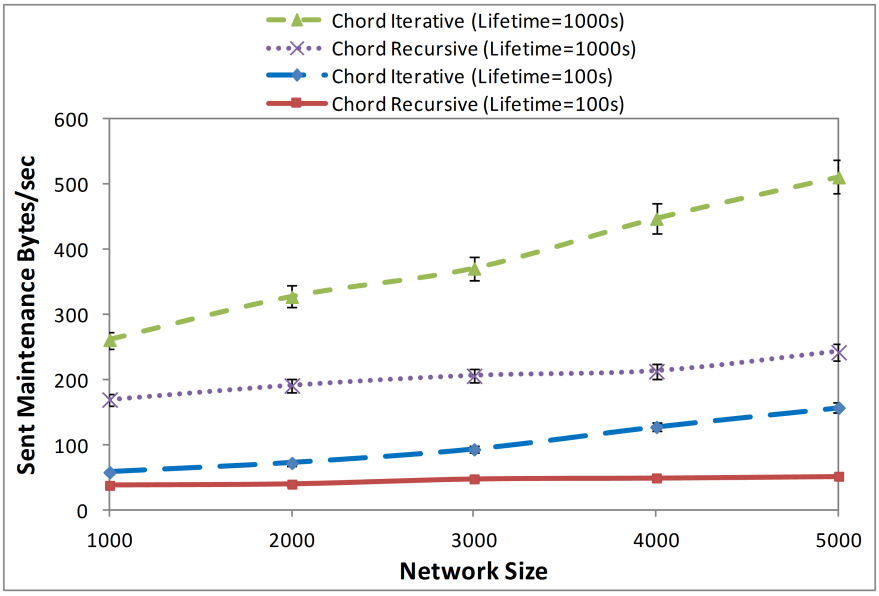



Figure 4.15: Chord: Sent maintenance bytes/s in iterative and recursive routing under NATed environment.

Figure 4.15 shows the effect of routing schemes on the bandwidth consumption with respect to two levels of churn. In this case, recursive routing brings benefit to the overlay as it requires less bandwidth than iterative routing. Recursive Chord requires 38 bytes/s to 58 bytes/s with network sizes from 1,000 nodes to 5,000 nodes in a high churn environment, whereas iterative Chord requires 59 bytes/s to 157 bytes/s with network sizes from 1,000 nodes to 5,000 nodes. In a medium churn scenario, recursive Chord requires 170 bytes/s to 242 bytes/s with network sizes from 1,000 nodes to 5,000 nodes, whereas iterative Chord requires 261 bytes/s to 520 bytes/s with network sizes from 1,000 nodes to 5,000 nodes. This is because recursive routing uses fewer messages for lookups compared to iterative routing [10]. Again, in a NATed environment, an UDP hole punching technique is used to establish a path between the NATed peers. With a recursive algorithm, a peer establishes a connection with only its neighbouring peers and has no need to establish any new connections as the responses are routed back through the same connections, whereas for an iterative algorithm, many more new connections need to be established as the initiating peer sends the query to its intermediate peer and the intermediate peer directly sends the response back to the querying peer. The initiating peer then sends a new request to another intermediate peer and this process continues until the target is found. It has been shown that iterative routing requires the establishment of $i-1$ new NAT connections (or holes), where $i$ is the total number of querying nodes in the network [10]. Therefore, using iterative routing in a NATed environment opens a potential security breach in the network.

The extra required bandwidth (Figure 4.15) in iterative routing is not that significant in comparison to recursive routing when the network size ranges from 1,000 to 5,000 nodes. However, the difference will be higher for a network consisting of a million nodes.

### 4.4.2 *Kademlia*

Kademlia has been simulated using iterative routing and recursive routing to investigate the effect of the routing approaches over NATs. The parameters for Kademlia using iterative and recursive routing are listed in Table 4.4. It should be noted that Kademlia does not use parallel RPCs in recursive routing.

Table 4.4: Simulation parameters for iterative and recursive routing in Kademlia.

| **Parameters** | **Iterative Routing** | **Recursive Routing** |
|---|---|---|
| Bucket Size, k | 8 | 8 |
| Siblings, s (nodes) | 2 | 2 |
| Bits per Digit, b (bits) | 1 | 1 |
| Lookup Redundant Nodes, r (nodes) | 8 | 8 |
| Bucket Refresh Interval (s) | 1,000 | 1,000 |
| No. of Parallel Lookups | 3 | 1 |

Figure 4.16 shows the lookup success ratio performance in iterative Kademlia and recursive Kademlia in a NATed environment. Kademlia with iterative routing has already shown its robustness under high churn environments. However, recursive Kademlia is not very efficient in a high churn environment. The plot shows that Kademlia with recursive routing has a 43% to 29% success ratio over 1,000 to 5,000 nodes in high churn. But in medium churn, it can achieve a satisfactory level of a 87% to 80% success ratio over 1,000 to 5,000 nodes. This difference is because of the nature of the routing mechanism in both approaches. In recursive routing, the intermediate nodes on the routing path directly forward the query to the node in the next hop, without sending any acknowledgement to the originator and the query can be routed as quickly as possible. If the lookup returns no response, the originator has to wait until a time out occurs as there is no knowledge of the actual cause. Hence, the originator has no control over the lookup process. On the contrary, iterative routing enables the originator to control the lookup process. In this routing, the originator sends a query to an intermediary node which replies directly to the originator with a closer location. The originator then sends a new request to the closer recommended node and the lookup process continues until the

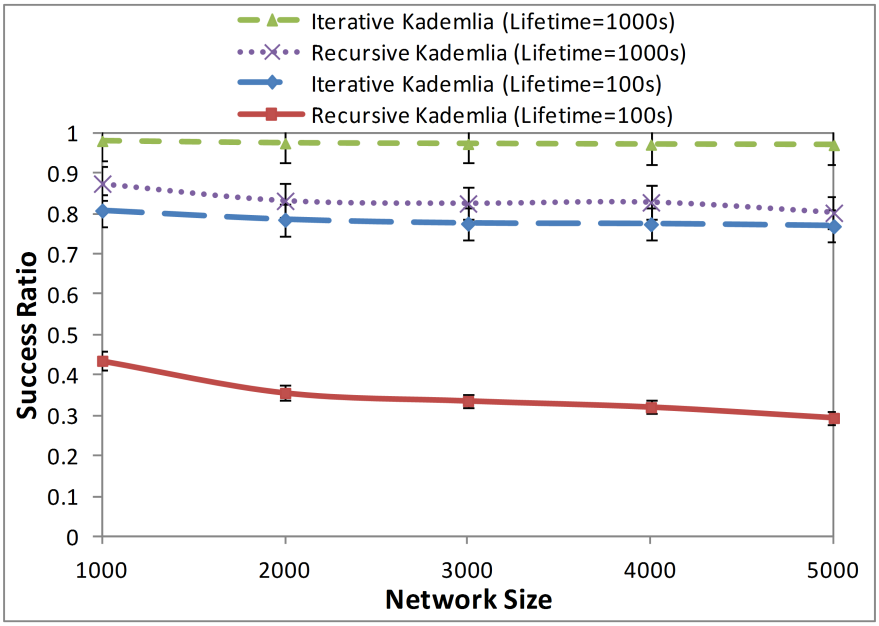



Figure 4.16: Kademlia: Lookup success ratio in iterative and recursive routing under NATed environment.

destination node is reached. Due to its manageable nature, iterative routing is more efficient for maintaining a successful lookup than recursive routing.

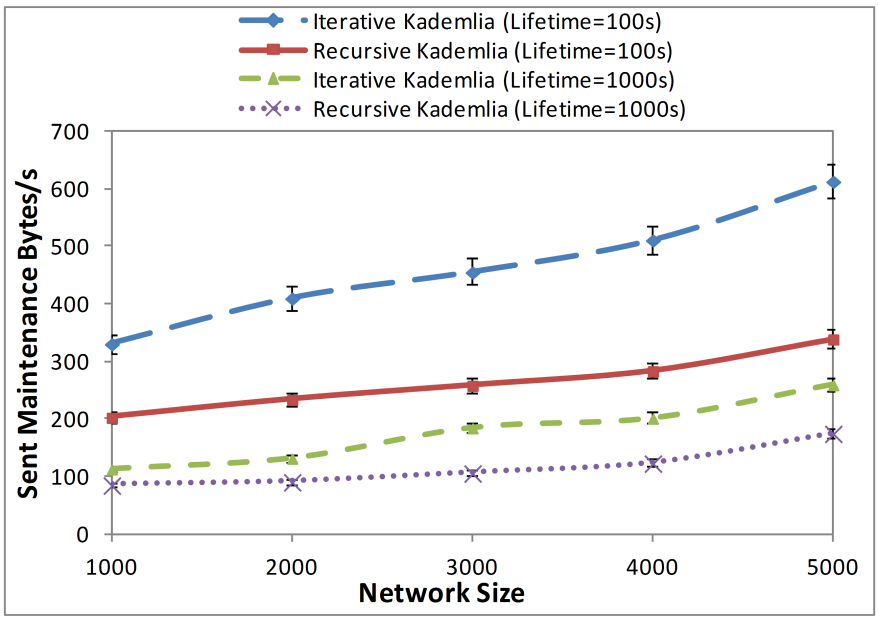



Figure 4.17: Kademlia: Sent maintenance bytes/s in iterative and recursive routing under NATed environment.

The bandwidth consumption is shown in Figure 4.17. Despite being in a NATed environment and comparing two different churn scenarios, recursive Kademlia exhibits a lower bandwidth consumption (~ 203 bytes/s to ~ 339 bytes/s in high churn and ~ 85 bytes/s to ~ 175 bytes/s in

medium churn over network size ranging from 1,000 to 5,000 nodes) than iterative Kademlia (~ 330 bytes/s to ~ 613 bytes/s in high churn and approximately ~ 113 bytes/s to ~ 259 bytes/s in medium churn over network size ranging from 1,000 to 5,000 nodes). Iterative Kademlia uses parallel RPCs (Remote Procedure Calls) which results in faster lookup but requires higher bandwidth than recursive Kademlia.

Again, in a NATed network, recursive routing does not require any new NAT hole to be punched, whereas iterative routing requires $i-1$ new NAT holes to be punched, where $i$ is the total number of querying nodes in the network, which includes the cost of these extra hole-punched connections in the cost of the routing.

### 4.4.3 *Discussion*

In this section, the performance of routing schemes has been evaluated as recursive and iterative routings have impacts over NATs. Evidently, in a network where NAT is a common feature, recursive routing is advantageous compared to iterative routing. However, iterative routing is more robust even in very high churn scenarios.

Chord and Kademlia have been simulated to investigate how the performance is affected using both routing approaches under NATs. In Chord, while using recursive routing, the success ratio is worse (approximately 65% to 35%) than in iterative routing (approximately 86% to 82%) in a medium churn scenario. But clearly it requires less bandwidth than iterative routing.

Kademlia, with iterative routing, is a stable overlay in a high churn environment and well suited to be used in mobile networks. In NATed and high churn environments, iterative Kademlia shows a reasonable success ratio (approximately 80% to 76%), and in a medium churn scenario environment, it exhibits almost a 98% to 97% success ratio, but it came with a higher cost of extra $i-1$ holes to be punched. On the contrary, recursive Kademlia came with an advantage that it does not require any extra new NAT holes to be punched and requires less bandwidth than iterative Kademlia. But it reveals a poor success ratio in a high churn environment (approximately 43% to 29% success ratio over 1,000 to 5,000 nodes).

In summary, a trade-off between iterative and recursive routing needs to be maintained in a churn intensive NATed network. If the network exhibits high churn, iterative routing is preferred, whereas in a NATed environment, recursive routing is useful. This observation motivates the need to investigate the potentiality of a hybrid routing mechanism to be used in a churn intensive NATed network such as a mobile network, which will be explored in Chapter 5.

This section outlines how the NAT functionalities using the proposed approach has been implemented in OverSim. Since the NAT functionality is implemented in the network layer, the configuration of the underlying network in OverSim needs to be modified. OverSim provides three different underlays: Simple, SingleHost and INET underlay (as discussed in Section 2.7.1). The underlying networks in Simple and SingleHost underlays are generated automatically using a fixed configuration file which offers a fixed topology and does not allow modifications of the underlying networks and hence, are unsuitable for implementing NAT functionalities. Only the INET underlay is suitable for implementing NAT functionalities. The INET underlay is based on the INET framework which offers the flexibility to configure the underlying network of any topology using the *NED (Network Descriptor)* language of OMNeT++. Furthermore, the well structured and extensively documented API enable easier development of newer modules offering additional capabilities. The underlay architecture is shown in Figure 4.18.

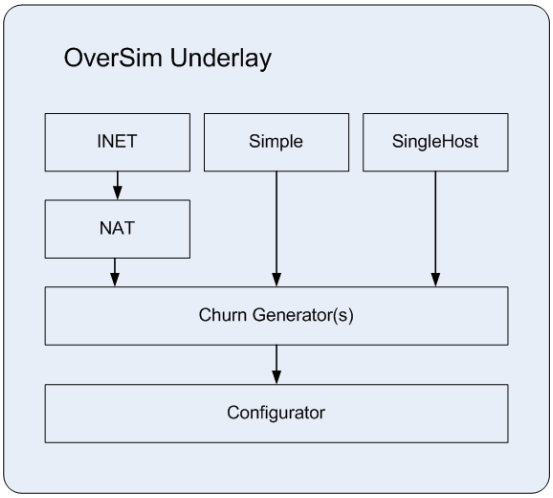



Figure 4.18: Underlay architecture of OverSim showing where NAT functionalities are implemented in INET underlay.

One such API introduced in the newer version of the INET framework is called the "IHook" API. It can be inserted within the link layer and network layer protocols inside any module and provides a hooking facility to perform additional tasks before a packet is transferred between these layers. This feature has been utilised to design a module that simulates the functionality of a NAT Router and performs network address translation for any (outgoing as well as incoming) network packet.

### 4.5.1 *Implementation Challenges*

- **Mismatch versions:** OverSim is based on an outdated version of INET (Version 1.99.2). The older version differs significantly from the newer version of INET (Version 2.2.0 onwards) in the way different modules interact with each other. Additionally, the older version lacks the extra functionalities (e.g. the IHook API) added in the newer version. The first challenge of the implementation is to integrate OverSim with the newer version of INET while keeping the functionalities of OverSim intact.
- **Novel NAT router module:** Since INET lacks the functionalities of NAT, a new module needs to be developed that can emulate the functionalities of a NAT router utilising the IHook API.
- **Network topology with NAT routers:** Next, it is required to design a topology to create a seamless connection of different private networks connected with different public networks using the newly developed NAT router module and existing router modules.
- **Novel network configurator:** OverSim uses a network configurator module that determines how and where in the network a new node will be added and how different packets between all nodes will be routed. It is required to design and develop a novel network configurator that will determine where (public or private networks) a new node will be added and how the packets between these heterogeneous networks will be routed.
- **Overlay modifications:** All overlay modules in OverSim are developed to function with nodes in public networks. The behaviour of all overlay modules needs to be changed so that they can handle requests from nodes residing in both public and private networks.

### 4.5.2 *Integration with OverSim*

Initially, OverSim has been integrated with an updated INET version. This introduced incompatibility in OverSim as many functionalities in OverSim were based on the outdated INET version which have been totally changed in the newer version. To resolve this incompatibility, the code-base of the OverSim has been modified by overwriting the erroneous functionalities with the updated code. This was a time consuming process in which it has been ensured that OverSim functions as intended.

To emulate the functionalities of a NAT router, a NAT router module has been developed by extending the existing router module with the additional capability of the IHook API. This enables the NAT router to perform network address translation for any packet originating from the private nodes attached to the respective NAT router and to perform reverse network address translation for any packet originating from any public node and is destined for any private node.

The network topology for the experiments is largely based on the topology illustrated in Figure 4.19. The number of private networks, private nodes and public nodes are controlled by parameters in the configuration file. A new network configurator has been developed that reads in these parameters and dynamically adds and removes private and public nodes in a random fashion, utilising the existing churn model. In addition, the global network configurator in OverSim is responsible for randomly selecting rendezvous servers for each private network and for providing interfaces to assign the corresponding rendezvous server for each newly joined private node. The network configurator is also responsible for enforcing the requirement that two rendezvous servers for the same private network cannot leave the network at the same time.

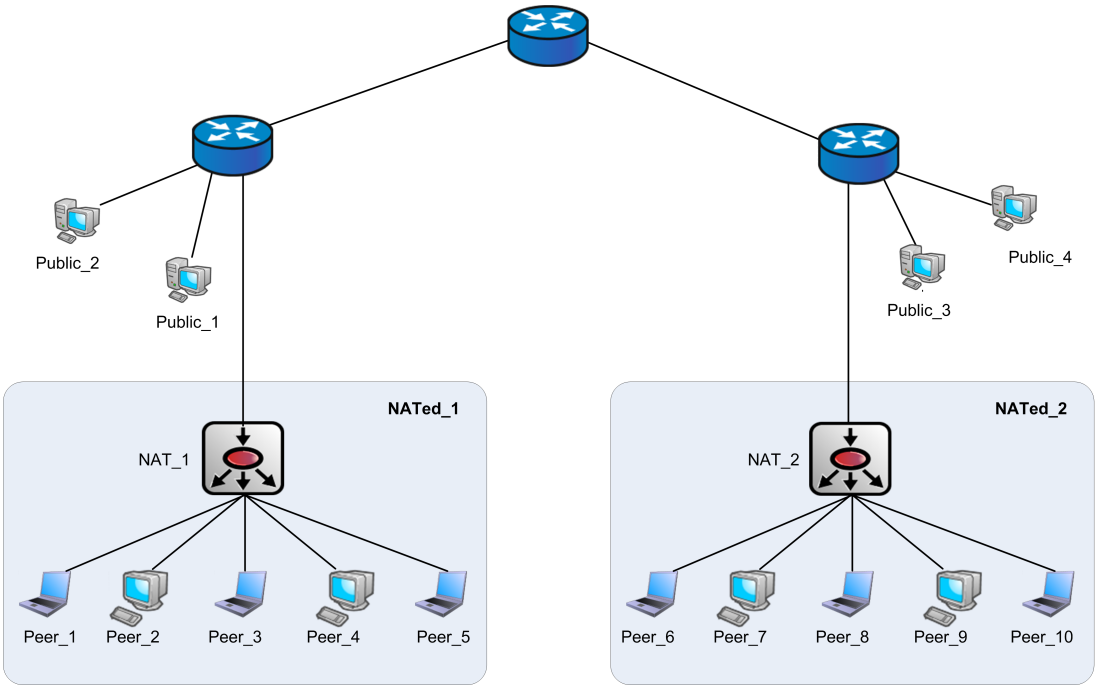



Figure 4.19: The network topology of a P2P system to implement UDP hole punching technique without a rendezvous server.

Next, all overlays in OverSim have been modified so that it can handle requests from public as well as private nodes. Additionally, the previous lookup method only relied on the inter-relation between the key and IP address of a node. It has been amended so that the inter-relation is based on the key and the IP address and port pair. This is required since all private nodes of a particular private network will share the same public IP address of the NAT router and hence, will be indistinguishable if only considered based on their IP addresses. All other functionalities of the overlay remain unchanged.

Finally, the code-base of Oversim has been amended to include the message flow discussed in Section 4.2.

The generic message flow within OverSim is as follows. As soon as a private node is added into a private network and initialised for joining the Overlay, it first goes through the registration phase. For this, it retrieves its respective rendezvous server from the network configurator and completes the registration message as discussed in Section 4.2. Once the registration process is complete, it joins the overlay following the protocol of the respective P2P overlay. Once joined, it needs to interact with its overlay neighbour(s). Similarly, the neighbours also need to communicate with it. This initiates a two-way communication protocol which is used to punch holes for the respective private peers using the mechanism discussed in Section 4.2. Once the holes are punched, the communication between the private peers are established and then the rest of the protocol for the overlay proceeds as intended.

The code will be shared with the OverSim community [1] in future, where it will be coordinated with the moderators to integrate it with the original OverSim code-base and release it along with the future version of OverSim.

## 4.6 SUMMARY

This chapter provided a performance evaluation of Chord, Kademlia and EpiChord with respect to NATed networks. The performance was validated by comparing the results with the public networks.

To simulate the private networks, NAT functionalities had been implemented in OverSim. To traverse the NATs, a NAT traversal approach, UDP hole punching without a rendezvous server, also was implemented. Section 4.2 described the UDP hole punching technique that had been used for the simulation considering two scenarios: peers behind a common NAT and peers behind different NATs.

Section 4.3 analysed the performance of Chord, EpiChord and Kademlia with two different churn levels under NATed and non-NATed environments. In NATed networks, the experimented overlays show similar performance in lookup success ratio and lookup hop count compared with the public networks. The only difference is in the bandwidth consumption. It is evident that all the three overlays require a higher bandwidth in NAT enabled networks than public networks.

Section 4.4 compared the performance of different routing algorithms over NATs. Only Chord and Kademlia had been investigated as EpiChord does not support a recursive algorithm. Both Chord and Kademlia using iterative routing exhibit better performance according to the success ratio in high churn environments. However, iterative routing brings a higher cost in bandwidth consumption. On the other hand, using recursive routing, Chord and Kademlia reduce the number of new connections that must be made through the NATs to enable P2P communications and offer advantages for traversing NATs.

Section 4.5 described how the NAT functionalities had been implemented in OverSim. Section 4.5.1 explained the implementation challenges whereas Section 4.5.2 illustrated how the updated INET framework had been integrated in OverSim to enable NAT functionalities.

The next chapter proposes an adaptive routing scheme combining recursive and iterative lookups considering a churn intensive NATed environment.

# 5

# CHURN AWARE ROUTING PROTOCOL (CHARP) IN NATED NETWORKS

*The previous chapter evaluated the performance of Chord, EpiChord and Kademlia in the presence of NATs. It also compared the performance of routing schemes under NATed environments. This chapter presents an adaptive routing protocol that switches between iterative and recursive routing techniques according to the current level of churn of a node in order to achieve an improved performance in NATed environments. In addition, the chapter also presents the results of experiments conducted utilising the proposed approach in order to show its applicability.*

## 5.1 INTRODUCTION

Existing P2P overlays use only a predefined routing protocol which is not changed dynamically. When a routing mechanism is decided at the design level, the lookup and maintenance operation is optimised to perform well in a particular scenario. However, the lookup operation may not be ideal under uncommon scenarios, such as churn intensive NATed scenarios. There may be a lookup mechanism that is highly effective in a particular scenario and not as effective in others. An adaptive algorithmic routing solution can detect such cases and smoothly switch between different routings to suit the current scenario.

The main motivation for an adaptive routing protocol is the experiment results, as presented in Chapter 4, which highlighted the fact that different routing protocols perform better (compared to each other) for different churn levels in NATed environments. Iterative routing is better for use in high churn environments, however, recursive routing is advantageous in NATed environments. Again, recursive routing reduces the number of total messages transmitted to the system and therefore, consumes less bandwidth than iterative routing. To traverse NAT and establish a P2P connection between peers, iterative routing needs to establish $i-1$ new connections (or holes) at NAT routers, where $i$ is the total number of querying nodes in the network. However, in recursive routing, a node establishes a hole with only neighbouring nodes and has no need to establish new holes on the way during the lookup process as responses are routed back through the same connections, established previously.

Based on these observations, an adaptive routing protocol can be the best solution to utilise both recursive and iterative routing with specific consideration to the high churn and low

bandwidth availability in mobile networks. In this chapter, such an adaptive routing protocol called ***Ch**urn **A**ware **R**outing **P**rotocol*, in short **ChARP**, is presented.

The main idea of the protocol is to switch between iterative and recursive routing techniques according to the current level of churn of a node. As a result, when the node is under a high churn scenario it will use the iterative routing, a preferred routing for churn intensive scenarios; whereas in low churn scenarios, it will switch its routing mechanism to recursive automatically. This chapter presents a novel churn aware routing scheme for DHT based P2P overlays. In the next section, the churn aware routing protocol that utilises this dynamic behaviour is proposed. Furthermore, the proposal is implemented in OverSim and then evaluated to investigate its applicability. This chapter also presents the results of experiments conducted using the proposed approach along with their analysis.

Section 5.2 describes the proposed adaptive routing protocol. Section 5.3 presents the performance evaluation of the proposed scheme and discusses its advantages over using only a single routing algorithm. Finally, Section 5.4 provides a brief summary of the chapter.

## 5.2 CHURN AWARE ROUTING PROTOCOL (CHARP)

### 5.2.1 *Overview and Design*

In the proposed routing protocol, it is assumed that the overlay consists of two types of nodes: NATed nodes and non-NATed nodes. As the focus of this research is on NATed churn intensive networks such as mobile networks, it is assumed that the majority of the nodes will be NATed mobile peers. The non-NATed nodes will have the global IP addresses and they can act as gateways (emulated rendezvous servers) to establish the direct P2P connection between the NATed nodes using a UDP hole punching NAT traversal technology as discussed in Chapter 4.

The proposed scheme is quite simple in nature. Its main highlight is a threshold (indicated by $\tau$) in the lifetime of a NATed peer which marks the time at which the respective peer makes a transition from one routing scheme to another. All public (non-NATed) peers will only utilise a pre-defined iterative routing protocol as this routing has already been proven to perform better in high churn scenarios. It does not matter how long they stay in a network, all public peers will only use iterative routing and therefore, no switching of routing mechanisms will occur in these peers. On the other hand, all NATed peers will initially utilise iterative routing. At regular intervals, the NATed peer will check if the lifetime of the peer reaches the threshold $\tau$. Once the threshold is reached, the respective peer will switch its routing mechanism to recursive routing.

The proposed routing scheme for a NATed environment is illustrated using a flowchart in Figure 5.1. The NATed peer is initiated with iterative routing (indicated by the *Routing* =

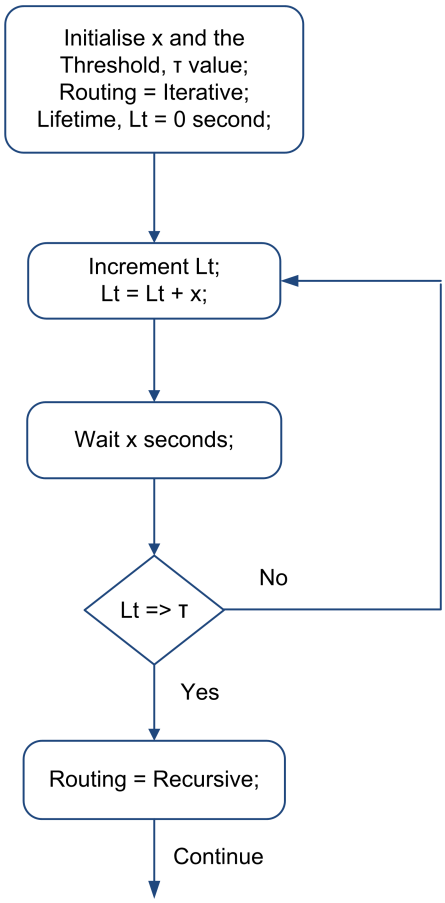



Figure 5.1: Flowchart of the proposed Churn Aware Routing Protocol (ChARP).

*Iterative* statement in Figure 5.1). The *lifetime* of a node starts at 0 seconds. An interval variable $x$ and the threshold variable $\tau$ are initialised with predefined values. After every $x$ seconds, the node checks if the lifetime of this node is greater than or equal to the threshold value. If no, the lifetime of the node is incremented by $x$ and waits for another $x$ seconds. If yes, the node changes its *Routing* mode iterative to recursive routing. This procedure continues until the lifetime of a node reaches the threshold value. Once a peer switches its routing scheme to recursive, it utilises that until it exits the P2P network.

In the next section, the performance of the proposed adaptive routing technique is evaluated and compared with the other two independent routing techniques.

## 5.3 Performance Analysis of the Churn Aware Routing Protocol (ChARP)

To investigate the applicability of ChARP, it has been implemented in OverSim, equipped with the same capabilities as discussed in Chapter 3 and 4. Different experiments have then been conducted using the chosen overlays. Throughout the evaluations in previous chapters, Kademlia and EpiChord have been shown to be the best performing P2P overlays that can be used in churn intensive mobile networks. In Chapter 3, using iterative routing, Kademlia

and EpiChord have shown better performance compared to Chord, Broose and Pastry under high churn. In Chapter 4, Kademlia and EpiChord also have been validated under NAT and it has been shown that the performance of these overlays is comparable in both NATed and non-NATed environments. That is why Kademlia and EpiChord were initially chosen as the candidate overlays for the proposed approach. Unfortunately EpiChord does not support recursive routing, and thus, the proposed routing scheme has been implemented only for Kademlia in OverSim. Therefore, it should be noted that the proposed routing protocol can be simulated in any P2P overlay, which supports both iterative and recursive routing.

### 5.3.1 *Experiment Setup*

The following setup has been used for the simulation experiment:

- ***NATed Environment:*** The simulation environment has been set up under two private NATed networks. The whole network consists of 5,000 nodes. Among the total number of nodes, 20% are public nodes (with public IP addresses) and the remaining 80% are private nodes equally divided in two private networks.
- P2P ***Overlay:*** ChARP has been implemented in Kademlia. The performance of Kademlia with iterative and recursive routing will also be compared with the proposed protocol.
- ***Churn:*** The level of churn is simulated by various node lifetime values ranging from 100 seconds to 1,500 seconds. A shorter node lifetime means a higher level of churn and vice versa.
- ***Churn Model:*** The simulations employ OverSim's Lifetime Churn model with Weibull distribution.
- ***Lookup Interval:*** Each node issues lookups for random keys at intervals exponentially distributed with a mean of 60 seconds.
- ***Transport Protocol:*** UDP has been used as a transport protocol in the simulation.
- ***Routing:*** Churn aware routing, iterative routing and recursive routing.
- ***Repetition:*** Each configuration has been repeated 5 times and 95% confidence intervals have been plotted.

### 5.3.2 *Evaluation Criteria*

The following two main metrics have been used to evaluate the performance from the simulation results:

1. ***Lookup Success Ratio:*** The total fraction of lookups for which the source node successfully receives the resource. This metric reveals the protocol's ability to deliver packets and to resolve the lookups. It also verifies its resilience under churn.

2. ***Mean Maintenance Traffic Load:*** The mean number of maintenance bytes sent per second by each node. This is considered as a parameter for bandwidth consumption.

### 5.3.3 *Configuration Parameters*

The configuration parameters for different routing in Kademlia are listed in Table 5.1. It should be noted that Kademlia does not send parallel lookups while using recursive routing.

Table 5.1: Simulation Parameters for different Routing Protocols in Kademlia.

| **Parameters** | **Iterative Routing** | **Recursive Routing** | **ChARP** |
|---|---|---|---|
| Bucket Size, $k$ | 8 | 8 | 8 |
| Siblings, $s$ (nodes) | 2 | 2 | 2 |
| Bits per Digit, $b$ (bits) | 1 | 1 | 1 |
| Lookup Redundant Nodes, $r$ (nodes) | 8 | 8 | 8 |
| Bucket Refresh Interval (s) | 1,000 | 1,000 | 1,000 |
| No. of Parallel Lookups | 3 | 1 | 3 (while iterative); 1 (while recursive) |

### 5.3.4 *Performance Evaluation*

ChARP has been implemented for Kademlia in OverSim to investigate the lookup performance. According to the protocol algorithm, all non-NATed peers have used only iterative routing, whereas the NATed peers have initially utilised iterative routing in Kademlia. When the lifetime of a NATed peer reaches the threshold value $\tau$, it switches to recursive routing. The threshold value for $\tau$ has been selected by conducting experiments using different node lifetime values as threshold values. From the results of these simulated experiments, a threshold value of 500 seconds seems to provide the optimal performance, and hence, has been considered for the analysis here. Using the threshold value, the lookup performance has been evaluated under a continuous level of churn in NATed environments. At the same time, the performance

of iterative and recursive routing has been compared and analysed with that of ChARP in Kademlia.

#### 5.3.4.1 *Effect of Churn*

To maintain low latency lookups and a higher lookup success ratio under churn, a peer must keep its routing state up-to-date. This section investigates the effect of ChARP on how a node copes with churn.

Figure 5.2 shows the lookup success ratio of all three routing schemes: iterative, recursive and ChARP for Kademlia under churn. In ChARP, when a node joins the Kademlia overlay, it starts with iterative routing and its lifetime starts from 0 seconds. After a certain time interval, it checks if the node lifetime increases (i.e. the level of churn minimises) to the threshold value $\tau$ (for Kademlia, $\tau = 500$, as discussed before). If it reaches the $\tau$ value, the node changes its routing method from iterative to recursive. From the simulation results plotted in the figure, it is found that iterative routing exhibits a better success ratio (approximately 80% to 98%) than ChARP (approximately 80% to 89%), whereas recursive routing shows the worst performance (approximately 36% to 75%) over the different level of node churn where the lifetime varies from 100 to 1,500 seconds. ChARP generates a similar success ratio as iterative routing until the node lifetime of 500 seconds. As ChARP enables a node to switch its routing mechanism from iterative to recursive at 500 seconds of lifetime, it is expected that its success ratio will decrease compared to the iterative routing. The interesting fact is that its performance has not decreased right away to the level of the recursive routing. This is because each node in the network remains in different stage of its lifetime and according to their individual lifetimes the routing mechanism also varies. As a result, the average combined overlay node lookup performance measures a better success ratio than recursive, however, lower than iterative.

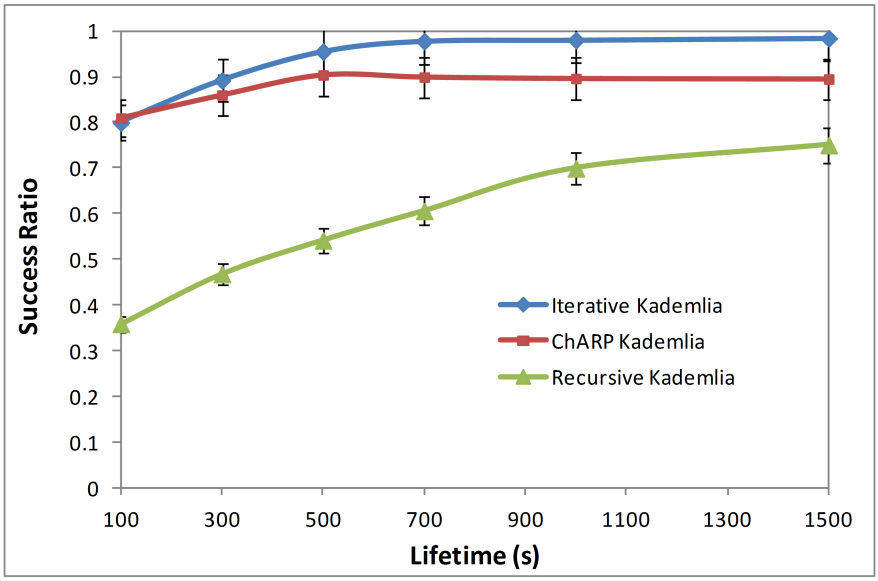



Figure 5.2: Lookup Success Ratio of Kademlia using Iterative, Recursive and ChARP.

#### 5.3.4.2 *Effect of Bandwidth Consumption*

The bandwidth consumption of Kademlia using iterative, recursive and ChARP has been plotted in Figure 5.3. Recursive Kademlia consumes less bandwidth (approximately 150 bytes/s to 71 bytes/s) than iterative (approximately 280 bytes/s to 98 bytes/s) and ChARP (approximately 255 bytes/s to 74 bytes/s) over high churn to low churn. Theoretically, ChARP should consume the bandwidth at the rate of iterative routing until the threshold $\tau$ value reaches at 500 seconds of node lifetime, and from 500 seconds onward, it should follow the similar rate as in recursive routing. But in the simulated experiment, it does not exhibit the expected result as the different node lifetimes are generated by using a probability distribution function (OverSim uses Weibull distribution in its churn model) and therefore, a certain portion of nodes use iterative whereas the rest of the nodes use recursive routing throughout the simulation time. As a result, the bandwidth consumption of the ChARP curve stays in the middle of the plot. Eventually, after a longer period of node lifetime, the ChARP curve coincides with the recursive routing curve (e.g. node lifetime of 1,500 seconds).

Clearly the proposed routing exhibits better results than using only a single iterative or recursive routing approach. It trades off between the two metrics, and has a considerably higher success ratio in high churn (as in iterative) and at the same time requires lower bandwidth in low churn (as in recursive).

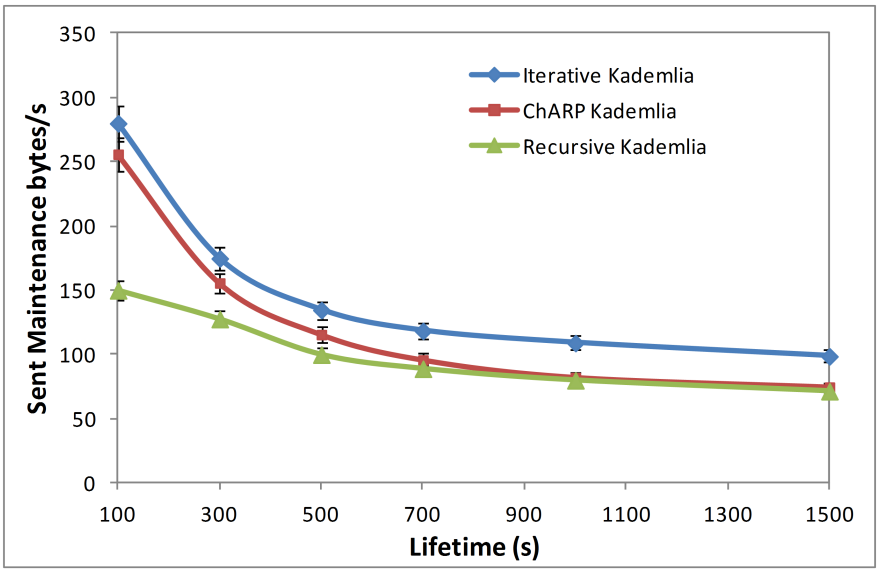



Figure 5.3: Sent Maintenance bytes/s of Kademlia using Iterative, Recursive and ChARP.

## 5.4 SUMMARY

This chapter has proposed and implemented a novel adaptive Churn Aware Routing Protocol (ChARP) for a P2P overlay to be used in churn intensive NATed mobile networks. Section 5.2

has presented the overview and design of the protocol. The proposed routing protocol in a P2P system has been defined in such a way that - it:

- utilises both routing techniques, iterative and recursive, on the basis of churn of a particular peer, instead of using one single routing method.
- gets the benefit of using two routing techniques together. Iterative routing is better in high churn scenarios, whereas recursive routing consumes less bandwidth. Therefore, using the proposed routing method, a P2P overlay can achieve an optimised result of both routing techniques.

As Kademlia supports both iterative and recursive routings, the proposed routing method has been implemented for Kademlia in OverSim. The performance of ChARP in Kademlia has been evaluated and the performance of the lookup success ratio and bandwidth consumption has been compared with iterative and recursive routings in Kademlia. For Kademlia, the threshold value to switch the routing protocol from iterative to recursive has been selected at 500 seconds. To achieve the desired trade-offs between lookup success ratio and bandwidth cost due to maintenance traffic,the threshold value can be selected differently in different P2P overlays. However, estimating tau is easy in simulation but more difficult in a real system.

By analysing the performance evaluation in Section 5.3, it has been shown that the proposed routing scheme is a good solution for a churn intensive NATed network such as a mobile network. Even though the proposed method has been implemented and analysed only in Kademlia, it is also suitable for any other P2P overlays that support iterative and recursive routing.

At the current state of the proposed scheme, each node switches the routing mechanism according to the current level of churn of a node. But it would be beneficial if the nodes could sense the churn rate of the whole network and therefore, can select the appropriate routing scheme depending on the situation. However, it could be quite challenging to enable a P2P system to sense the churn state of the whole network in real application scenarios.

# Part III

# CONCLUSIONS

# 6

# CONCLUSIONS

*This chapter concludes the thesis. Firstly, the key contributions of this work have been highlighted. Following this, the limitations have been identified. The chapter also gives an outlook on future work opened up by the research performed in this thesis.*

## 6.1 INTRODUCTION

DHT-based structured P2P overlay networks offer an efficient routing architecture that is adaptive, self-organising, fault tolerant, scalable and massively distributed. Such an overlay network provides a suitable platform for developing distributed applications such as information retrieval, multi-player gaming, internet telephony, media distribution and distributed computing. It would be desirable to also offer such P2P applications on devices connected to mobile networks. Nevertheless there are concerns regarding the performance and efficiency of these overlays in mobile environments.

Therefore, the aim of this thesis has been to investigate the performance of a number of structured P2P overlays under conditions commonly experienced in mobile data networks. A number of multi-hop and one-hop P2P overlays have been evaluated under extreme conditions found in mobile networks such as high node churn, low bandwidth availability and NAT restrictions. Based on the experimentation results, a novel and improved routing protocol has been proposed and implemented in a structured P2P overlay. In addition, an overview of the content of this thesis as a whole is provided at the end of this chapter.

## 6.2 CONTRIBUTIONS

This section lists the main contributions and the key finding of the thesis.

1. **Determining the most suitable configuration of a number of structured P2P overlays for churn intensive mobile networks.** A P2P overlay has a number of different configuration parameters such as stabilisation delay, routing table size and levels of parallelism, that affect the performance of the overlay. The first contribution of this thesis is to identify the most suitable parameter set for a number of structured P2P overlays in a particular scenario. The parameters of four multi-hop structured P2P overlays: Chord, Pastry, Kademlia and Broose, and one single-hop structured P2P overlay: EpiChord have been evaluated in the presence of high churn in the OverSim P2P framework and the

best performing configuration has been selected to be used in churn intensive mobile networks on the basis of maximum lookup success ratio and minimum bandwidth consumption.

2. **Performance analysis of a number of structured P2P overlays under churn.** Utilising the chosen set of parameters, the most suitable overlays have been finalised for use in mobile networks. For this, a comparative analysis has been performed with each P2P overlay, considering the restrictions found in mobile networks such as high churn and the relatively low bandwidth availability. The simulation results suggest that Kademlia and EpiChord are the two most appropriate P2P overlays to be implemented with mobile networks according to lookup success ratio under high levels of churn.

3. **Implementation of a NAT traversal mechanism in OverSim.** To simulate NAT functionalities, private networks have been integrated with public networks in OverSim. To traverse NAT, the UDP hole punching technique without a dedicated rendezvous server has been implemented. Instead, different public peers have been selected to emulate rendezvous servers. Using this setup, different P2P overlays have been simulated and validated under two scenarios: peers behind a common NAT and peers behind different NATs.

4. **Performance evaluation and validation of a number of structured P2P overlays under NATed and non-NATed networks.** The performance of Chord, Kademlia and EpiChord has been evaluated under churn intensive NATed networks. The performance has also been validated by comparing the results with the public non-NATed networks. The validation shows that the results are comparable in both NATed and non-NATed environments.

5. **Performance analysis of iterative and recursive routing methods in NATed environments.** The presence of NATs in a network influences the routing techniques used in P2P overlays. Recursive routing is more resilient to IP connectivity restrictions posed by NATs, but not very robust in high churn environments. On the contrary, iterative routing is more suitable to high churn networks, but difficult to use in NATed environments. Therefore, the performance impact of iterative and recursive routings on Chord and Kademlia has been analysed in the presence of different levels of churn in NATed environments. Both Chord and Kademlia using iterative routing exhibit better performance according to the success ratio in high churn environments. However, iterative routing requires a higher bandwidth consumption. On the other hand, using recursive routing, Chord and Kademlia reduce the number of new connections that must be made through the NATs to enable P2P communications and hence, the recursive routing is advantageous for NAT traversal.

6. **A novel Churn Aware Routing Protocol.** Based on the insights gained from the performance study of existing P2P overlays under churn and NAT restrictions, a novel Churn Aware Routing Protocol (ChARP) has been proposed and implemented for Kademlia in OverSim. The main idea of the protocol is to switch between iterative and recursive routing techniques according to the current level of churn of a node. Experimental results established using the proposed approach exhibit an improved performance in comparison to any single (iterative or recursive) routing approach, considering a trade-off between lookup success ratio and bandwidth consumption.

## 6.3 LIMITATIONS

- **Simulation-based study.** This thesis has provided an extensive simulation using a well-known P2P simulator OverSim by simulating a network containing thousands of nodes. The results of all experiments have been verified against other researchers' results (when available). However, it must be noted that it is a simulation exercise and a real implementation of this approach would be useful.
- **Reliance on public peers.** ChARP needs to rely on public peers for emulating the functionalities of rendezvous servers. The higher is the number of such emulated servers, theoretically, the better it is for distribution of loads. Currently such nodes are randomly chosen. It would be useful to select such nodes based on their probable lifetime so that the nodes which have better probability to have a long lifetime should be chosen as emulated servers.
- **Incompatibility with symmetric NAT.** The current NAT traversal approach, UDP hole punching, using ChARP is incompatible with symmetric NAT.

## 6.4 FUTURE DIRECTIONS

There are a number of possible future directions for research inspired by this thesis. A few such possible future directions are highlighted next.

1) **Effect of increasing the number of emulated servers:** Currently the UDP hole punching technique in OverSim has been implemented by ensuring that there will be always two emulated servers for each private network throughout the simulation. If a private network consists of a large number of nodes, it will be unrealistic to burden the chosen emulated servers with the additional communication overhead required for the UDP hole punching technique. Introducing more emulated servers will reduce the probability of all rendezvous servers leaving the network simultaneously and will also minimise the burden on the emulated servers. However, this may introduce unprecedented side

effects on the performance of the P2P overlay. Investigating how the number of emulated servers can be increased and how it will affect the performance of the corresponding P2P overlay is a possible future research direction.

2) **Increasing the number of NATs:** To simulate a real world application scenario, the number of NATs needs to be increased. For this, the preferred method will be to generate the network topology dynamically by reading the network parameters from a configuration file that specifies the desired number of private networks using NATs. This is a challenging task that will require changing the way a network is configured dynamically in OverSim. Furthermore, the effect of increasing private networks as well as NATs on the performance of a P2P overlay must be investigated.

3) **Network-wide churn awareness:** The proposed dynamic routing method, ChARP, has been implemented by enabling each private node to switch its routing mechanism after the node lifetime reaches a threshold. A better approach would be to achieve a network-wide consensus on the state of the current level of churn and then activate a network-wide switching of the routing mechanism in each peer. It will be interesting to investigate how such a network-wide awareness of churn can be achieved, how a signal for switching the routing mechanism can be propagated in all existing peers and what will be the effect on the system performance in such a setup.

4) **Energy efficiency measurement:** There has been a tremendous advancement in mobile phones. However, modern technology is still far away from providing a sustainable energy efficient mobile phone, even smartphones suffer from extremely limited battery power. The scope of this thesis did not include energy constraint in terms of peer's battery life. Mobile peers' collaboration regarding energy saving is potential future research work.

5) **Impact of varying the ratio of public nodes and private nodes:** In the current implementation of NATed environments, the number of public nodes is only 20% of the total number of nodes. Another possible future research direction would be to investigate the impact of varying the ratio of public and private nodes and analysing its effect on the performance of the P2P overlay.

6) **Integrating symmetric NAT.** Another possible future work would be to investigate how a symmetric NAT can be integrated within the implemented NAT traversal approach.

## 6.5 SUMMARY

Chapter 1 *"Introduction"* provided the motivation and benefits of deploying DHT-based P2P overlays on mobile networks. It also discussed the importance of investigating the effect of

different limitations imposed by mobile networks on existing popular structured P2P overlays. Accordingly, the key objectives of this thesis were identified.

Chapter 2 *"Background and Related Work"* provided a brief background on DHT-based structured P2P overlays: Chord, Kademlia, Pastry, Broose and EpiChord. It also listed a number of challenges to the adoption of P2P in mobile and wireless networks. Among the challenges mentioned, this thesis focused on high churn rate, data consumption and NAT restriction. In addition, it discussed a number of available techniques for NAT traversal. Moreover, this chapter presented a number of novel approaches to implement P2P overlays in wireless cellular networks along with their pros and cons. It was found that there is not a single approach available that can solve the problems of high churn, energy consumption and NAT restrictions. This chapter also gave a brief review of a number of available P2P simulators and justified why OverSim was selected for the experimentations in this thesis.

Chapter 3 *"Performance Evaluation of Structured* P2P *Overlays under Churn"* evaluated the performance of Chord, Kademlia, Pastry, Broose and EpiChord, and determined the best configuration parameters to achieve optimal performance under churn using the OverSim P2P framework. After selecting the best parameters, each overlay was compared with each other to identify the most suitable overlay to be used in churn intensive mobile networks. The simulation results suggest that Kademlia and EpiChord are the two most appropriate P2P overlays to be implemented on the mobile network according to the lookup success ratio under high levels of churn.

Chapter 4 *"Performance Evaluation of Structured* P2P *Overlays under NAT"* outlined how the NAT functionalities had been implemented in OverSim. To traverse the NAT, a NAT traversal approach, UDP hole punching without a rendezvous server was incorporated. Next, the performance of Chord, EpiChord and Kademlia was evaluated under NATed networks and then validated with their performance with non-NATed networks. In addition, this chapter evaluated the impact of iterative and recursive routing for Chord and Kademlia under NATs .

Chapter 5 *"Churn Aware Routing Protocol in NATed Networks"* presented an adaptive Churn Aware Routing Protocol (ChARP) that switches between iterative and recursive routing techniques according to the current level of churn of a node, in order to achieve an improved performance over a fixed (iterative or recursive) routing method in NATed environments. In addition, the chapter presented the results of conducted experiments utilising the proposed approach in order to validate its applicability.

Part IV

# BACK MATTER

# REFERENCES


[1] OverSim: The Overlay Simulation Framework. URL `http://oversim.org/`.

[2] I. Baumgart and B. Heep. Fast but economical: A simulative comparison of structured peer-to-peer systems. In *Proceedings of the 8th Euro-NF Conference on Next Generation Internet (NGI 2012)*. IEEE, June 2012.

[3] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, pages 79–84, May 2007.

[4] Bobby Bhattacharjee, Seungjoon Lee, Ruggero Morselli, Rob Sherwood, Dave Levin, and Suman Banerjee. NICE. URL `http://www.cs.umd.edu/projects/nice/`.

[5] Andrew Biggadike, Daniel Ferullo, Geoffrey Wilson, and Adrian Perrig. NATBLASTER: Establishing TCP connections between hosts behind NATs. In *ACM SIGCOMM Asia Workshop*, volume 5, 2005.

[6] Bittorrent. BitTorrent. Accessed on 23 October, 2015. URL `http://www.bittorrent.com/`.

[7] Martin Blunn. A peer-to-peer network framework utilising the public mobile telephone network. *M.Phil Thesis, Dept. of Computing Science and Mathematics, University of Stirling, Scotland*, 2011.

[8] Mohamed Boucadair, Reinaldo Penno, and Dan Wing. Universal Plug and Play (UPnP) Internet Gateway Device-Port Control Protocol Interworking Function (IGD-PCP IWF). 2013.

[9] Alan Brown and Mario Kolberg. Tools for Peer-to-Peer Network Simulation. *draftirtf-p2prf-core-simulators-00. txt*, 2006.

[10] David A Bryan, Marcia Zangrilli, and Bruce B Lowekamp. Challenges of DHT design for a public communications system. *College of William & Mary Computer Science Department Technical Report WMCS200603*, 2006.

[11] Miguel Castro, Peter Druschel, Y Charlie Hu, and Antony Rowstron. Exploiting network proximity in peer-to-peer overlay networks. Technical report, Citeseer, 2002.

[12] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony IT Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489–1499, 2002.

[13] C-F Michael Chan and S-H Gary Chan. Distributed Hash Tables: Design and Applications. In *Handbook of Peer-to-Peer Networking*, pages 257–280. Springer, 2010.

[14] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making Gnutella-like P2P systems scalable. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418. ACM, 2003.

[15] Farida Chowdhury and Mario Kolberg. Performance Evaluation of EpiChord under High Churn. In *Proceedings of the 8th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, pages 29–36. ACM, 2013.

[16] Bram Cohen. Bittorrent Protocol Specification. Accessed on 1 March, 2012, February 2008. URL `http://bittorrent.org/beps/bep_0003.html`.

[17] James Cowie, A Ogielski, and D Nicol. The SSFNet Network Simulator. *Software on-line: http://www. ssfnet. org/homePage. html*, 2002.

[18] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a common API for structured peer-to-peer overlays. In *Peer-to-Peer Systems II*, pages 33–44. Springer, 2003.

[19] Krishna Dhara, Yang Guo, Mario Kolberg, and Xiaotao Wu. Overview of Structured Peer-to-Peer Overlay Algorithms. In Xuemin Shen, Heather Yu, John Buford, and Mursalin Akon, editors, *Handbook of Peer-to-Peer Networking*, pages 223–256. Springer US, 2010. ISBN 978-0-387-09751-0.

[20] Peter Druschel and Antony Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 75–80. IEEE, 2001.

[21] Kjeld Egevang and Paul Francis. The IP Network Address Translator (NAT). Technical report, 1994.

[22] eMule Project. Accessed on 23 October, 2015. URL `http://www.emule-project.net/home/perl/general.cgi?l=1`.

[23] George C Engelmayr, Mingyu Cheng, Christopher J Bettinger, Jeffrey T Borenstein, Robert Langer, and Lisa E Freed. Accordion-like honeycombs for tissue engineering of cardiac anisotropy. *Nature materials*, 7(12):1003–1010, 2008.

[24] Ericsson. Ericsson Mobility Report, 2015. Accessed on 10 October 2015. URL `http://www.ericsson.com/mobility-report`.

[25] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-Peer Communication Across Network Address Translators. In *USENIX Annual Technical Conference, General Track*, pages 179–192, 2005.

[26] A-T Gai and Laurent Viennot. Broose: a practical distributed hashtable based on the de-bruijn topology. In *Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on*, pages 167–174. IEEE, 2004.

[27] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[28] Pedro García, Carles Pairot, Rubén Mondéjar, Jordi Pujol, Helio Tejedor, and Robert Rallo. *Planetsim: A new overlay network simulation framework*. Springer, 2005.

[29] Chris GauthierDickey, Virginia Lo, and Daniel Zappala. Using n-trees for scalable event ordering in peer-to-peer games. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 87–92. ACM, 2005.

[30] Thomer Gil, Frans Kaashoek, Jinyang Li, Robert Morris, and Jeremy Stribling. P2Psim, A Simulator for Peer-to-Peer Protocols, 2003.

[31] Saikat Guha, Yutaka Takeda, and Paul Francis. NUTSS: A SIP-based approach to UDP and TCP network connectivity. In *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 43–48. ACM, 2004.

[32] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues. Efficient routing for peer-to-peer overlays. In *NSDI*, volume 4, pages 9–9, 2004.

[33] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert Van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Peer-to-Peer Systems II*, pages 160–169. Springer, 2003.

[34] Selim Gurun, Priya Nagpurkar, and Ben Y. Zhao. Energy consumption and conservation in mobile peer-to-peer systems. In *Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*, MobiShare '06, pages 18–23, New York, NY, USA, 2006. ACM. ISBN 1-59593-558-4. doi: 10.1145/1161252.1161258. URL `http://doi.acm.org/10.1145/1161252.1161258`.

[35] Jani Hautakorpi and Gonzalo Camarillo. Evaluation of DHTs from the viewpoint of interpersonal communications. In *Proceedings of the 6th international conference on Mobile and Ubiquitous Multimedia*, pages 74–83. ACM, 2007.

[36] Dave Hoch. Time in App Increases by 21% Across All Apps. Accessed on 21 November 2014. URL `http://info.localytics.com/blog/time-in-app-increases-by-21-across-all-apps`.

[37] Quirin Hofstätter, Stefan Zoels, Maximilian Michel, Zoran Despotovic, and Wolfgang Kellerer. Chordella - a hierarchical peer-to-peer overlay implementation for heterogeneous, mobile environments. *Peer-to-Peer Computing, IEEE International Conference on*, 0: 75–76, 2008. doi: http://doi.ieeecomputersociety.org/10.1109/P2P.2008.42.

[38] Teerawat Issariyakul and Ekram Hossain. *Introduction to network simulator NS2*. Springer, 2011.

[39] M Frans Kaashoek and David R Karger. Koorde: A simple degree-optimal distributed hash table. In *Peer-to-peer systems II*, pages 98–107. Springer, 2003.

[40] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

[41] I. Kelényi and J.K. Nurminen. Energy Aspects of Peer Cooperation Measurements with a Mobile DHT System. In *Communications Workshops, 2008. ICC Workshops '08. IEEE International Conference on*, pages 164 –168, may 2008. doi: 10.1109/ICCW.2008.36.

[42] Imre Kelényi. Mobile Kademlia for Symbian OS. Budapest University of Technology, Dept. of Automation and Applied Informatics, Applied Mobile Research Group. URL `http://www2.aut.bme.hu/Portal/MobileDHT.aspx?lang=en`.

[43] Imre Kelényi and Jukka K. Nurminen. Optimizing Energy Consumption of Mobile Nodes in Heterogeneous Kademlia-Based Distributed Hash Tables. In *Proceedings of the 2008 The Second International Conference on Next Generation Mobile Applications, Services, and Technologies*, NGMAST '08, pages 70–75, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3333-9. doi: 10.1109/NGMAST.2008.59. URL `http://dx.doi.org/10.1109/NGMAST.2008.59`.

[44] Imre Kelényi, Jukka Nurminen, Ákos Ludányi, and Tamas Lukovszki. Modeling resource constrained bittorrent proxies for energy efficient mobile content sharing. *Peer-to-Peer Networking and Applications*, 5:163–177, 2012. ISSN 1936-6442. URL `http://dx.doi.org/10.1007/s12083-011-0107-5`. 10.1007/s12083-011-0107-5.

[45] Mario Kolberg, Florence Kolberg, Alan Brown, and John Buford. A Markov model for the Epichord peer-to-peer overlay in an XCAST enabled network. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 1935–1942. IEEE, 2007.

[46] Gerald Kunzmann. Recursive or iterative routing? Hybrid! In *KiVS Kurzbeiträge und Workshop*, pages 189–192, 2005.

[47] Ben Leong, Barbara Liskov, and Erik D. Demaine. EpiChord: Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management. Technical Report MIT-LCS-TR-963, MIT, Cambridge, MA, August 2004.

[48] Jinyang Li, Jeremy Stribling, Robert Morris, M Frans Kaashoek, and Thomer M Gil. A performance vs. cost framework for evaluating DHT design tradeoffs under churn. In *INFOCOM 2005: 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 1, pages 225–236. IEEE, 2005.

[49] Jian Liang, Rakesh Kumar, and Keith W Ross. The FastTrack overlay: A measurement study. *Computer Networks*, 50(6):842–858, 2006.

[50] Rohan Mahy, Philip Matthews, and Jonathan Rosenberg. Traversal using relays around NAT (TURN): relay extensions to session traversal utilities for NAT (STUN). *Internet Request for Comments*, 2010.

[51] MaidSafe. The MaidSafe Platform. URL `http://www.maidsafe.net/distributed-network-features.html`.

[52] Gurmeet Singh Manku, Mayank Bawa, Prabhakar Raghavan, et al. Symphony: Distributed Hashing in a Small World. In *USENIX Symposium on Internet Technologies and Systems*, page 10, 2003.

[53] Petar Maymounkov and David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, 2002. Springer-Verlag. ISBN 3-540-44179-4.

[54] Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *P2P 2009. IEEE 9th International Conference on Peer-to-Peer Computing, 2009*, pages 99–100. IEEE, 2009.

[55] Ruggero Morselli, Bobby Bhattacharjee, Michael Marsh, Aravind Srinivasan, et al. Efficient lookup on unstructured topologies. *Selected Areas in Communications, IEEE Journal on*, 25 (1):62–72, 2007.

[56] Stephen Naicken, Anirban Basu, Barnaby Livingston, and Sethalat Rodhetbhai. A survey of peer-to-peer network simulators. In *Proceedings of The Seventh Annual Postgraduate Symposium, Liverpool, UK*, volume 2, 2006.

[57] NS-3. NS-3 network simulator. Accessed on 10 October, 2015. URL `https://www.nsnam.org/`.

[58] J.K. Nurminen and J. Noyranen. Energy-Consumption in Mobile Peer-to-Peer - Quantitative Results from File Sharing. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 729 –733, jan. 2008. doi: 10.1109/ccnc08.2007.167.

[59] Zhonghong Ou, Erkki Harjula, Otso Kassinen, and Mika Ylianttila. Feasibility evaluation of a communication-oriented P2P system in mobile environments. In *Proceedings of the 6th International Conference on Mobile Technology, Application and Systems*, Mobility '09, pages 43:1–43:8, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-536-9. doi: 10.1145/1710035.1710078. URL http://doi.acm.org/10.1145/1710035.1710078.

[60] Zhonghong Ou, Erkki Harjula, Otso Kassinen, and Mika Ylianttila. Performance evaluation of a Kademlia-based communication-oriented P2P system under churn. *Computer Networks*, pages 689–705, 2010.

[61] Krzysztof Pawlikowski, H-DJ Jeong, and J-SR Lee. On credibility of simulation studies of telecommunication networks. *Communications Magazine, IEEE*, 40(1):132–139, 2002.

[62] Himabindu Pucha, Saumitra M Das, and Y Charlie Hu. How to implement DHTs in mobile ad hoc networks. In *MobiCom 2004: Proceedings of the 10th ACM International Conference on Mobile Computing and Network*, 2004.

[63] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-addressable Network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 161–172, New York, NY, USA, 2001. ACM. ISBN 1-58113-411-8. doi: 10.1145/383059.383072. URL http://doi.acm.org/10.1145/383059.383072.

[64] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling Churn in a DHT. In *In Proceedings of the USENIX Annual Technical Conference*, 2004.

[65] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99–100. IEEE, 2001.

[66] Jonathan Rosenberg, Joel Weinberger, Christian Huitema, and Rohan Mahy. STUN-simple traversal of User Datagram Protocol (UDP) through Network Address Translators (NATs). Technical report, RFC 3489, IETF, Mar, 2003.

[67] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[68] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 188–201. ACM, 2001.

[69] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Networked group communication*, pages 30–43. Springer, 2001.

[70] Kazuyuki Shudo, Yoshio Tanaka, and Satoshi Sekiguchi. Overlay weaver: An overlay construction toolkit. *Computer Communications*, pages 402–412, 2008.

[71] Spyros Sioutas, George Papaloukopoulos, Evangelos Sakkopoulos, Kostas Tsichlas, and Yannis Manolopoulos. A novel distributed P2P Simulator Architecture: D-P2P-Sim. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 2069–2070. ACM, 2009.

[72] Pyda Srisuresh, George Tsirtsis, Praveen Akkiraju, and Andy Heffernan. DNS extensions to Network Address Translators (DNS_ALG). 1999.

[73] Dominik Stingl, Christian Gross, Julius Ruckert, Leonhard Nobach, Aleksandra Kovacevic, and Ralf Steinmetz. Peerfactsim. kom: A simulation framework for peer-to-peer systems. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 577–584. IEEE, 2011.

[74] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM. ISBN 1-58113-411-8. doi: 10.1145/383059.383071.

[75] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202. ACM, 2006.

[76] Di Wu, Ye Tian, and Kam-Wing Ng. Analytical Study on Improving DHT Lookup Performance under Churn. In *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, P2P '06, pages 249–258, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2679-9. doi: 10.1109/P2P.2006.4. URL `http://dx.doi.org/10.1109/P2P.2006.4`.

[77] Pinggai Yang, Jun Li, Jun Zhang, Hai Jiang, Yi Sun, and Eryk Dutkiewicz. SMBR: A novel NAT traversal mechanism for structured Peer-to-Peer communications. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 535–539. IEEE, 2010.

[78] C. Huiyou Z. Tengyue, L. Liao. The Mobile Node ID Kademlia for Multimedia Stream Transmission in Wireless Network. In *4th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4, 2008.

[79] Ben Y Zhao, Ling Huang, Jeremy Stribling, Sean C Rhea, Anthony D Joseph, and John D Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41–53, 2004.

[80] Stefan Zöls, Rüdiger Schollmeier, Wolfgang Kellerer, and Anthony Tarlano. The Hybrid Chord Protocol: A Peer-to-Peer Lookup Service for Context-Aware Mobile Applications. In Pascal Lorenz and Petre Dini, editors, *ICN (2)*, volume 3421 of *Lecture Notes in Computer Science*, pages 781–792. Springer, 2005. ISBN 3-540-25338-6. URL `http://dblp.uni-trier.de/db/conf/icn/icn2005-2.html#ZolsSKT05`.

[81] Stefan Zöls, Zoran Despotovic, and Wolfgang Kellerer. Cost-Based Analysis of Hierarchical DHT Design. In Alberto Montresor, Adam Wierzbicki, and Nahid Shahmehri, editors, *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P 2006), 2-4 October 2006, Cambridge, United Kingdom*, pages 233–239. IEEE Computer Society, 2006. ISBN 0-7695-2679-9. doi: http://doi.ieeecomputersociety.org/10.1109/P2P.2006.13.

[82] Stefan Zöls, Simon Schubert, Wolfgang Kellerer, and Zoran Despotovic. Hybrid DHT Design for Mobile Environments. In *Agents and Peer-to-Peer Computing*, pages 19–30. Springer, 2006.

[83] Stefan Zöls, Zoran Despotovic, and Wolfgang Kellerer. Load balancing in a hierarchical DHT-based P2P system. In *Proceedings of the 3rd International Conference on Collaborative Computing: Networking, Applications and Worksharing, White Plains, New York, USA, November 12-15, 2007*, pages 353–361. IEEE, 2007. ISBN 1-4244-1317-6. doi: http://dx.doi.org/10.1109/COLCOM.2007.4553855.

[84] Stefan Zöls, Quirin Hofstätter, Zoran Despotovic, and Wolfgang Kellerer. Achieving and Maintaining Cost-Optimal Operation of a Hierarchical DHT System. In *ICC*, pages 1–6, 2009.

[85] Mohammad Zulhasnine, Changcheng Huang, and Anand Srinivasan. Towards an effective integration of cellular users to the structured Peer-to-Peer network. *Peer-to-Peer Networking and Applications*, 5:178–192, 2012. ISSN 1936-6442.