DT-Chord: Implementing Chord over Delay Tolerant Network

by

Rakib Uddin Ahmed

Submitted to

Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
MASTER OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DHAKA-1000

May, 2011

M. Engg. Project Report

DT-Chord: Implementing Chord over Delay Tolerant Network

by

Rakib Uddin Ahmed

Submitted to

Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Engineering in Computer Science and Engineering

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka-1000

May, 2011

The thesis titled '**DT-Chord: Implementing Chord over Delay Tolerant Network'**, submitted by Rakib Uddin Ahmed, Roll No. 100705001P, Session: October, 2007, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Engineering in Computer Science and Engineering and approved as to its style and contents. Examination was held on May 02, 2011.

Board of Examiners

1. _____
   Dr. Reaz Ahmed — Chairman (Supervisor)
   Associate Professor
   Department of Computer Science and Engineering
   BUET, Dhaka-1000.

2. _____
   Dr. Md. Monirul Islam — Member (Ex-officio)
   Professor & Head
   Department of Computer Science and Engineering
   BUET, Dhaka-1000.

3. _____
   Dr. Md. Humayun Kabir — Member
   Associate Professor
   Department of Computer Science and Engineering
   BUET, Dhaka-1000.

4. _____
   Dr. Mahmuda Naznin — Member
   Associate Professor
   Department of Computer Science and Engineering
   BUET, Dhaka-1000.

Candidate's Declaration

This is to certify that the work entitled 'DT-Chord: Implementing Chord over Delay Tolerant Network' is the outcome of the research carried out by me under the supervision of Dr. Reaz Ahmed in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka-1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

_____

Rakib Uddin Ahmed

Candidate

To

My Beloved Mother

-without whom I can't think of me

Contents

List of Figures

List of Tables

Acknowledgement

All praises to Allah, the most benevolent and merciful

I am extremely grateful for the chance to have worked with my advisor, Dr. Reaz Ahmed. He has provided me enormous help and encouragement throughout this work. His intellect, rigor, enthusiasm will always inspire me.

I want to thank the other members of my thesis committee: Dr. Md. Monirul Islam, Dr. Md. Humayun Kabir, and Dr. Mahmuda Naznin for their valuable suggestions.

I cannot forget my friend, Md. Moshiur Rahman for his support.

Finally, I would like to express my gratitude to my mother for her continuous support.

May almighty Allah rewards them all.

Abstract

Delay Tolerant Networks (DTNs) are a class of networks designed to address several challenging connectivity issues such as sparse connectivity, long or variable delay, intermittent connectivity, asymmetric data rate, high latency, high error rates and even no end-to-end connectivity. The DTN architecture adopts a store-and-forward paradigm and a common bundle layer located on the top of region-specific network protocols in order to provide interoperability of heterogeneous networks (regions). In this type of network, a source node originates a message (bundle) that is forwarded to an intermediate node (fixed or mobile) thought to be more close to the destination node. The intermediate node stores the message and carries it while a contact is not available. Then the process is repeated, so the message will be relayed hop by hop until reaching its destination. A fundamental problem that confronts future applications of DTN is how to efficiently locate the DTN node that stores a particular data item. In this case, flooding search seems to be the only method. However, this usually results in so-called broadcast storm problem, which leads to significant performance degradation in DTN.

Distributed hash table (DHT) based protocols provide near-optimum data lookup time for resolving queries made on large P2P network. Lookup latency is important to applications that use DHTs to locate data. In order to achieve low latency lookups, each node needs to consume bandwidth to keep its routing tables up to date under churn. This thesis presents DT-Chord- a DHT based application protocol for Delay Tolerant Network that minimizes delay while locating data in DTN. This thesis also shows performance comparisons between base Chord and DT-Chord protocols over Delay Tolerant Network. Finally, we provide a methodology to determine the relative importance of tuning application protocol parameters under different workloads and network conditions.

Chapter 1

1. Introduction

Delay-tolerant networks (DTN) [1] are a promising new development in network research, that offer the hope of connecting people and devices that hitherto were either unable to communicate, or could do so only at great cost. For example, today it is possible to connect from a cell phone to millions of powerful servers around the world. As successful as these networks have been, they still cannot reach everywhere, and for some applications their cost is prohibitive. The reason for these limitations is that current networking technology relies on a set of fundamental assumptions that are not true in all environments. The first and most important assumption is that an end-to-end connection exists from the source to the destination, possibly via multiple intermediaries. This assumption can be easily violated due to mobility, power saving, or unreliable networks. For example, if a wireless device is out of range of the network (*e.g.* the nearest cell tower, 802.11 base station, *etc.*), it cannot use any application that requires network communication. Delay-tolerant networking is an attempt to extend the reach of networks. It promises to enable communication between "challenged" networks, which includes deep space networks [2], data MULEs [3], underwater networks [4], wildlife tracking sensor networks like ZebraNet [5], and opportunistic mobile ad-hoc networks [6]. The core idea is that these networks can be connected if protocols are designed to accommodate disconnection.

As an example of where these networks are useful, consider a classroom where each student has a laptop, but there is no network infrastructure. One would like the students to collaborate on projects using the wireless network cards in the laptops, and also to communicate with the Internet. Delay-tolerant networking can make this happen, as illustrated in Figure 1.1. The laptops communicate with each other to exchange data. If the destination laptop is not present, which may occur if the student has gone home, the

network stores the messages until they return. To communicate with the Internet, the school could be serviced via a router attached to a bus traveling between the school and an Internet gateway. This device picks up requests from the school and delivers them to the gateway, and then provides the responses on its next trip. First Mile Solutions sells a system called DakNet that is based on this idea [7], while the Wizzy Digital Courier Project uses a simple one-hop delay tolerant network to provide Internet access to rural South African schools [8].

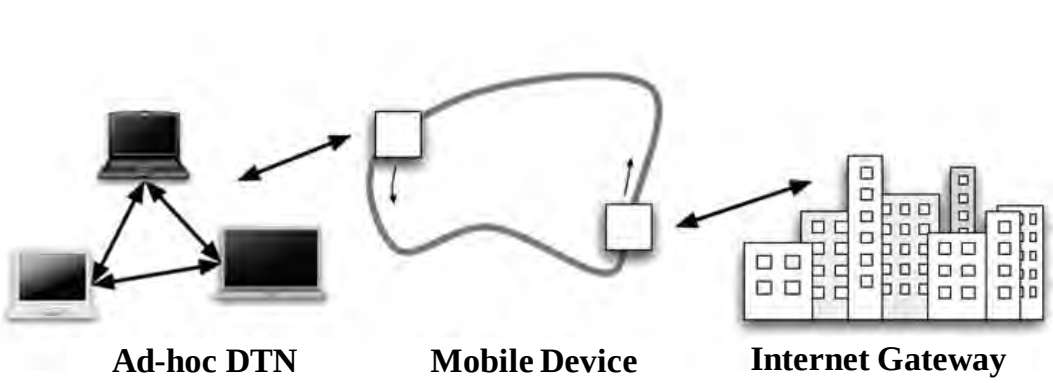


Figure 1.1: Laptops communicating with each other and the Internet via delay-tolerant networking

There are many other applications for delay-tolerant networks. In developing regions, applications range from education to health care to government services [9]. In developed nations, researchers have proposed augmenting low bandwidth Internet connections with a high-bandwidth delay tolerant network built by sending physical media, such as DVDs, through the postal system [10]. This allows very large files to be quickly and cheaply exchanged while small files and control messages are exchanged over a low bandwidth link. Others have investigated using DTNs to provide Internet access to cars, by connecting temporarily to roadside wireless base stations [11]. DTNs could also be used to gather data from everything ranging from sensors in oceans [12], to satellites in space [3].

Peer-to-Peer (P2P) technology offers alternative communication architecture from traditional client-server architecture. Nowadays, the most popular usage of P2P technology is file sharing among users. In a P2P file-sharing network, users can publish

and share their own possessions with others which are currently online. P2P users can also download interesting files from others. Unstructured or less-structure P2P protocols, as their names suggested, have no or little restriction on the structure of the overlay. Hence, it is difficult for users to locate the resources they need in an unstructured P2P network. In such cases, flooding search seems to be the only method to locate an object. However, this usually results in so-called broadcast storm problem, which leads to significant performance degradation in DTN. On the other hand, structured P2P protocols, such as Chord [14], Pastry [15], Tapestry [16] and CAN [17], are designed for locating objects effectively in wired networks. These structured protocols form a fully organized overlay with the proposed protocols. Logarithmic complexity is achievable for key operations such as joining, maintaining, or querying in the P2P overlay. Due to the attractive low overhead, researchers paid lots of attention on Chord [13, 18, 19, 20, 21, 22]. Researchers strived to enhance the overlay performance and to apply it to large scale deployment. Nevertheless, these P2P protocols usually assume reliable communication among nodes. This is true in wired network but difficult to achieve in Delay Tolerant Network.

Chord [14] is one of the popular structured P2P protocols. Chord is usually deployed on application layer as a P2P overlay. Chord behaves elegantly because of the simplicity of the design and scalability to construct large-scale overlay network with low complexity. In a N-node Chord overlay, each node only have to keep contact just $O(logN)$ nodes, and a query can be done within $O(logN)$ steps. Furthermore, nodes only need to maintain few local information (successor and predecessor) in order to guarantee the consistency of Chord. However, previous work has addressed some problems and challenges to Chord operating in wireless and mobile networking environments. One problem is Chord overlay can efficiently operate in stable networks where churn rate is low. In highly dynamic networks like DTN, the routing table maintenance cost is very high. The communication overhead for maintaining a virtual overlay network in a challenged environment and the performance penalty arising from routing in this virtual overlay are deemed to be main issues.

1.1 Problem Definition

A fundamental problem that confronts future applications of DTN is how to efficiently locate the DTN node that stores a particular data item. In this case, flooding search seems to be the only method. It is robust, but asking every node is very expensive. In worst case, O(*N*) messages are required per lookup . Asking only some nodes might not find key. The obvious downside of flooding is that it generates loads of unnecessary traffic that consumes resources that may be scarce in a DTN (*e.g.*, bandwidth, power, storage capacity). It is known that distributed hash table (DHT) based Internet peer-to-peer (P2P) protocols provide near-optimum data lookup times for queries made on networks of distributed nodes. In DHT based lookup protocol, the efficiency measure the number of application-level hops taken on the path. However, the true efficiency measure is the end-to-end latency of the path. Because the nodes could be geographically dispersed, some of these application-level hops could involve transcontinental links, and others merely trips across a LAN; routing algorithms that ignore the latencies of individual hops are likely to result in high latency paths. A generic mapping of these protocols to delay tolerant network is, however perceived as difficult.

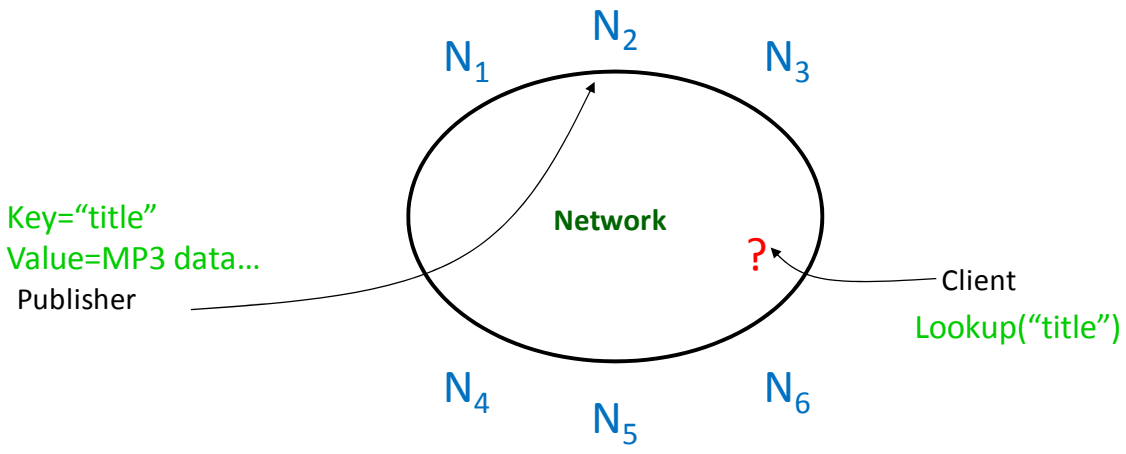


Figure 1.2: A scenario for understanding the lookup problem

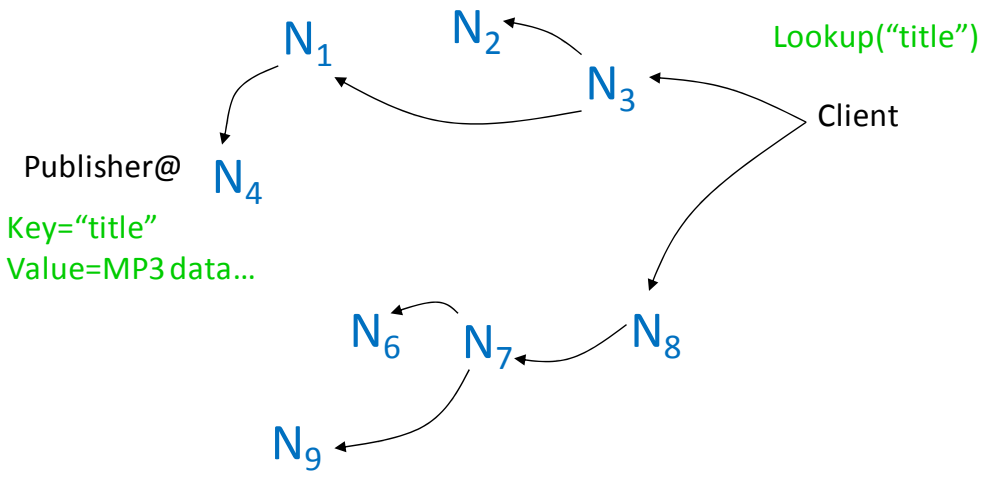


Figure 1.3: Flooded queries to find data

1.2 Motivation

Retrieving information from the Web has become part of our daily lives. In many applications such as distributed wikis or photo sharing, users need to be able to find content. In order do bring these applications to the domain of DTNs, a lookup scheme is required that works despite the challenging network conditions. A search engine helps by mapping lookup queries to resource locators that are likely to point to content including the sought information. However, there are scenarios in which communicating with a search engine is not an appealing or even suitable option. Such situations occur, for example, when the desired information may be available from the vicinity via a local wireless (ad-hoc) network and a connection to the search engine is unavailable or not reliable. In this case, a search mechanism is desirable that allows directly querying other nodes in an unreliable networking environment.

The traditional approach to map lookup queries to resources is implemented by maintaining an index about the resources. The index contains keywords and maps them to sets of related resources. This can be used to map a query consisting of several keywords to a list of suggested lookup results. Examples of centralized search engines using such indices are Google, MSN, and Yahoo. The problem with such centralized services is that

it is expensive to keep the index up-to-date and the assumption is made that the clients can reach both the index and the locations where the contents reside. There are approaches for managing decentralized indexes in peer-to-peer environments. We investigate how to perform lookup of contents in delay-tolerant networks using distributed and decentralized index.

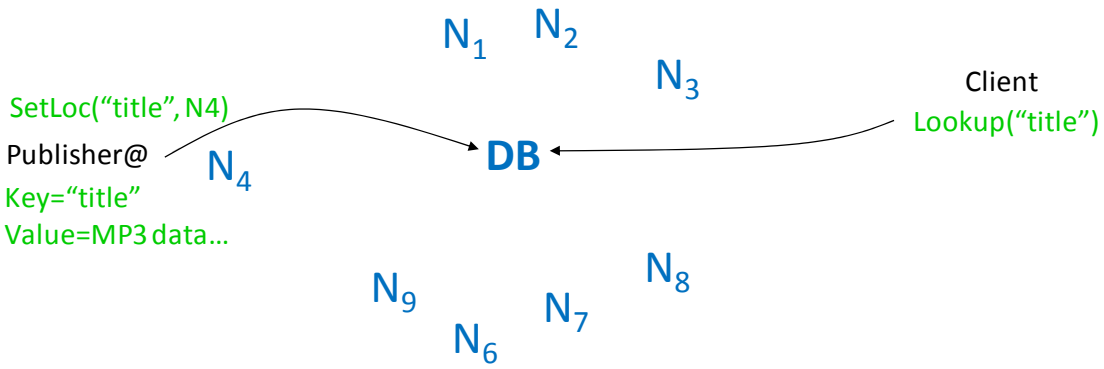


Figure 1.4: Centralized lookup

1.3 Contributions

The contributions of this thesis are:

1. A DHT based lookup protocol, DT-Chord that minimizes delay while locating data in DTN.

2. A study for understanding the relationship between communication costs and the resulting performance benefits.

3. A comparative performance study to determine the relative importance of tuning protocol parameters under different workloads and network conditions.

Apart from the above contributions, this thesis presents a different style of designing distributed protocols. We argue that a distributed protocol over DTN should be conscious of its communication overhead in order to stay robust across a wide range of operating environments. We demonstrate how to design such a cost aware protocol that uses the bandwidth resource sparingly and efficiently.

1.4 Organization

The focus of this thesis is to implement Chord over Delay Tolerant Network. The rest of this project is organized as follows: the first chapter describes the motivation and goal of the thesis.

Chapter 2: Background

Chapter 2 is divided into two parts. First part describes distributed hash table (DHT) based peer-to-peer (P2P) lookup protocol, Chord. Second part of this chapter provides a brief introduction of Delay Tolerant Network Architecture.

Chapter 3: Proposed System: Delay Tolerant Chord

Chapter 3 describes the design and implementation of DT-Chord, a DHT based lookup protocol for delay tolerant network. We explain the design principles of DT-Chord's neighbor selection and routing algorithm followed by the protocol details.

Chapter 4: Performance Evaluation

Chapter 4 presents a comparative performance evaluation of DT-Chord against Chord. We present the simulation results of DT-Chord implementation over DTN along with brief discussion on the simulator P2PSim [23] and experimental dataset.

Chapter 5: Performance Tuning of DT-Chord

Chapter 5 explores the parameter space to find DT-Chord's latency vs. bandwidth tradeoff and compare the efficiencies of different design choices. We evaluate how efficiently different design choices use additional bandwidth for better lookup performance.

Chapter 6: Conclusion and Future Research

Concluding remarks are presented in Chapter 6. This chapter also presents our future research goals and possible research.

Chapter 2

Background

2.1 Distributed Hash Table (DHT)

A hash-table interface is an attractive foundation for a distributed lookup algorithm because it places few constraints on the structure of keys or the data they name. The main requirements are that data be identified using unique numeric keys, and that nodes be willing to store keys for each other. This organization is in contrast to Napster and Gnutella, which search for keywords, and assume that data is primarily stored on the publisher's node. However, such systems could still benefit from a distributed hash table—for example, Napster's centralized database recording the mapping between nodes and songs could be replaced by a distributed hash table. A DHT implements just one operation: lookup(key) yields the identity (*e.g.*, IP address) of the node currently responsible for the given key. A simple distributed storage application might use this interface as follows. Someone who wants to publish a file under a particular unique name would convert the name to a numeric key using an ordinary hash function such as SHA-1, then call lookup(key). The publisher would send the file to be stored at the resulting node. Someone wishing to read that file would obtain its name, convert it to a key, call lookup(key), and ask the resulting node for a copy of the file. A complete storage system would have to take care of replication, caching, authentication, and other issues; these are outside the immediate scope of the lookup problem.

2.2 Chord Protocol

Chord [14] assigns ID's to both keys and nodes from the same one-dimensional ID space. Chord Each Chord node has a unique *m* bit node identifier (ID), obtained by hashing the

node's IP address. Chord views the IDs as occupying a circular identifier space. Keys are also mapped into this ID space, by hashing them to *m*-bit key IDs. The node responsible for a key is the node with the identifier which most closely follows the key in the circular key space; we refer to this node as the successor of *k* and to the several nodes after *k* as the successor list of *k*. In Figure 2.2, the successor of key K10 is node N17. Note that N9 (the predecessor of the key) maintains pointer to the node N17 and can definitively return K10's successor. Chord maintains a routing table of log*N* pointers to other nodes in the system and can resolve a mapping by sending log*N* messages, where *N* is the number of nodes in the system. Because Chord keeps a small amount of state, it is able to maintain the state efficiently in large or unstable systems.

2.2.1 Chord Lookup Algorithm

A Chord node uses two data structures to perform lookups: a successor list and a finger table. Only the successor list is required for correctness, so Chord is careful to maintain its accuracy. The finger table accelerates lookups, but does not need to be accurate, so Chord is less aggressive about maintaining it. The following discussion first describes how to perform correct (but slow) lookups with the successor list, and then describes how to accelerate them up with the finger table. Every Chord node maintains a list of the identities and IP addresses of its *r* immediate successors on the Chord ring. The fact that every node knows its own successor means that a node can always process a lookup correctly: if the desired key is between the node and its successor, the latter node is the key's successor; otherwise the lookup can be forwarded to the successor, which moves the lookup strictly closer to its destination.

A new node *n* learns of its successors when it first joins the Chord ring, by asking an existing node to perform a lookup for *n*'s successor; *n* then asks that successor for its successor list. The *r* entries in the list provide fault tolerance: if a node's immediate successor does not respond, the node can substitute the second entry in its successor list. All *r* successors would have to simultaneously fail in order to disrupt the Chord ring, an event that can be made very improbable with modest values of *r*. An implementation

should use a fixed r, chosen to be $2\log_2 N$ for the foreseeable maximum number of nodes N.

Lookups performed only with successor lists would require an average of N/2 message exchanges, where N is the number of servers. To reduce the number of messages required to $O(\log N)$, each node maintains a finger table with m entries. The i^{th} entry in the table at node n contains the identity of the *first* node that succeeds n by at least 2^i $(0 \leq i < m)$ on the ID circle. Thus every node knows the identities of nodes at power-of-two intervals on the ID circle from its own position. A new node initializes its finger table by querying an existing node. Existing nodes whose finger table or successor list entries should refer to the new node find out about it by periodic lookups.

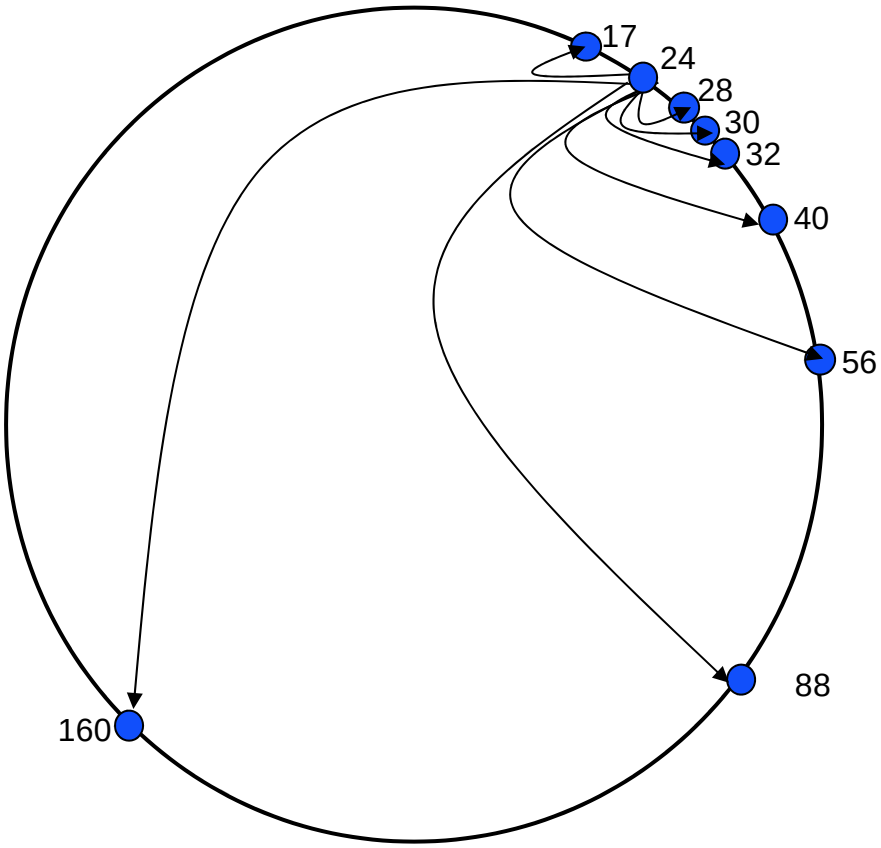


Figure 2.1: A Chord node's finger table. Each node maintains logN pointers to other nodes. The pointers are spaced exponentially around the ring (i.e. ½, ¼, ⅛ … of the way around the ring). In the example above, node N24's routing table is shown. The most distant finger of node 24 points to the first node that is more than half-way around the ring (after 24 + 256/2 = 152); this is node N160 in the example. This spacing allows a lookup to halve the distance to the target at each step. Lookups complete in log$N$ time.

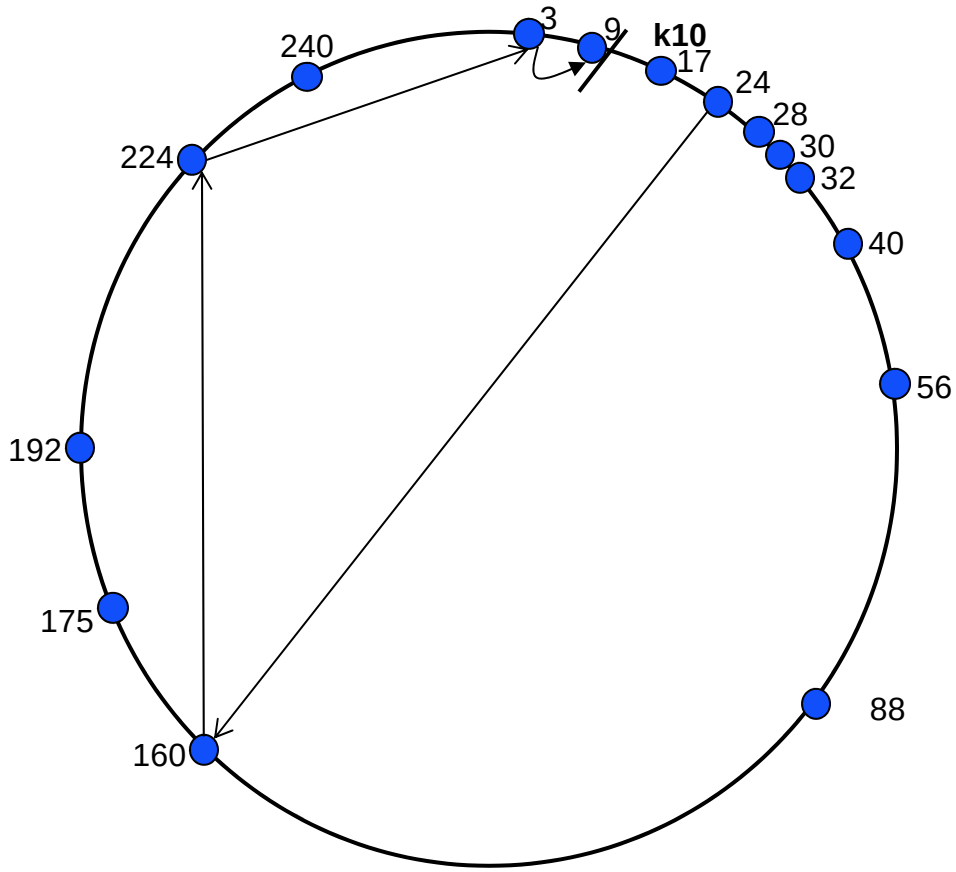


Figure 2.2: The path of a lookup for key K10 originated at node N24.

As shown in Figure 2.2, to find a key's successor using the finger table, a node routes the request greedily in ID space. At each step, the requesting node consults the node that is closest to the key's ID but still precedes the key on the ring. That node replies with the identity of the best next hop node. Eventually the requesting node will contact the key's predecessor, which will return its successor list: the answer to the lookup query. Because of the power-of-two distribution of a node's finger table, the node will always have a pointer that is at least half of the distance to the key at each step in the lookup. Because the distance remaining to the key is halved at each step of the lookup, we expect a lookup to require $O(\log N)$ hops. Figure 2.2 shows the path taken by an example lookup. The base Chord algorithm is iterative. In iterative lookup (Figure 2.3) intermediate nodes send information about possible next hop nodes to the originating before the lookup can proceed.

```
// Ask node n to find id's successor; first
// finds id's predecessor, then asks that
// predecessor for its own successor.
n.find_successor(id)
        n´ = find_predecessor(id);
        return n´.successor list();

// Ask node n to find id's predecessor.
n.find predecessor(id)
        n´ =n;
        while (id = ∉ (n´, n´.successor()])
                l = n´.closest_predecessor_list(id);
                n´= max n″∈l   s.t. n″ is alive
        return n´;

// Ask node n for the node in its finger table or
// successor list that most closely precedes id.
n.closest_predecessor_list(id)
return  { n´ ∈{fingers U successors} s.t. n´ ∈ (n, id]}
```

Figure 2.3: The pseudo-code to find the successor node of an identifier *id* using iterative lookup. Remote procedure calls are preceded by the remote node.

2.3 Delay Tolerant Network

The ability to transport, or route, data from a source to a destination is a fundamental ability all communication networks must have. Delay tolerant networks (DTNs), are characterized by their lack of connectivity, resulting in a lack of instantaneous end-to-end paths. A delay-tolerant network is an overlay on top of a number of diverse regional networks, including the Internet. Within a DTN, the regional networks may be extremely

remote in terms of delay, and may employ, for example, different wireless technologies. The DTN overlay accommodates these varying network characteristics and provides a service that works regardless of “difficult” conditions in the underlying networks.

The motivation for DTN is that in certain situations the protocols used in the internet simply do not work. Examples of such situations are partitioned networks, highly asymmetric data rates, high error rates and long delays. A typical use-case for a DTN is an interplanetary network, *e.g.* a satellite orbiting Earth communicating with another satellite orbiting Mars.

DTN works by introducing a new protocol layer, the bundle layer, on top of the transport layer (Figure: 2.4). The transport protocols used in the underlying regional networks need not be the same – the bundle layer is the glue that binds all the various lower layers together. The applications in the DTN only need to communicate with the homogenous bundle layer. We emphasize that our goal is to implement Chord over DTN at the application level. So, we left the detailed discussion of other protocol layers of DTN.

Bundles are messages that consist of the bundle header, control information (provided by the source application for the destination application) and user data. In essence, a bundle just extends the data encapsulation hierarchy with one further level. The bundle layer has a set of mechanisms to overcome the difficulties of intermittent, long delay networks. The basic idea is to use store-and-forward message switching, *i.e.* hold bundles in a persistent storage along the communication path until the next hop comes available. An end-to-end path need not exist when the bundle is initially sent. Also, the bundle layer protocol is non-conversational in the sense that the nodes communicate between each other using simple sessions with minimal or no round-trips.

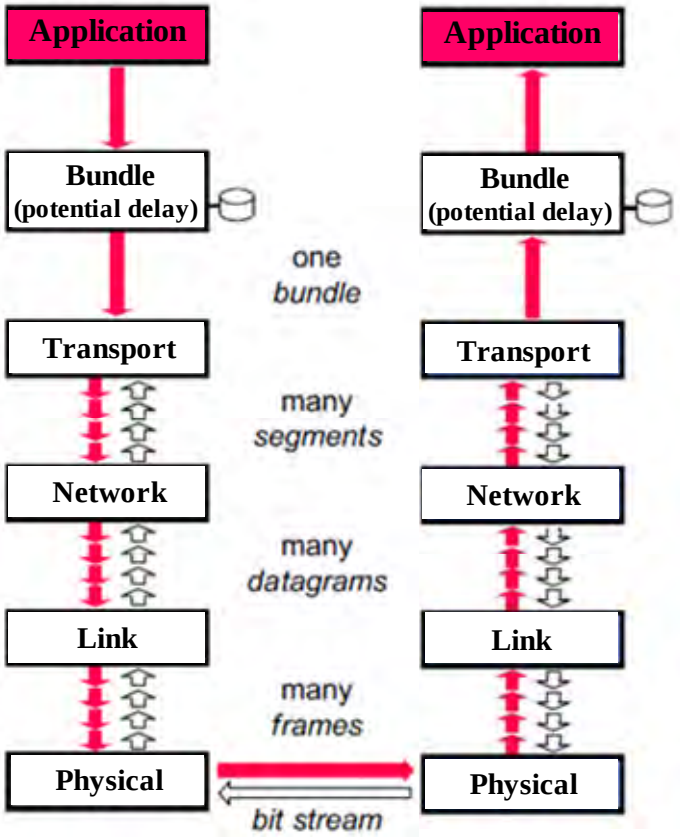


Figure 2.4: DTN Protocol Stack

Acknowledgments from the receiving node are optional. To cope with long delays while still allowing TCP (or some other conversational protocol) to be used as the underlying protocol in some parts of the network, the bundle layer utilizes transport-layer termination. This means that a DTN node acts as a surrogate for a TCP end-node, isolating the TCP connection from the bundle layer.

2.4 Summary

Distributed hash table (DHT) based peer-to-peer (P2P) protocols provide near-optimum data lookup time for resolving queries made on large P2P network. In a DTN, an end-to-end path may not be available at all times and path latency may be comparatively large. So, application protocols need to tolerate the delay resulting from the environmental challenges.

Chapter 3

Proposed System: DT-Chord

3.1 Overview

We address the problem of efficient resource retrieval in challenged scenarios. Distributed Hash Tables (DHTs) organize the peer-to-peer network in a structured manner to provide a hash-table-like lookup interface. Use of traditional P2P approaches proposed for reliable and connected wireless networks does not always show effectiveness in challenged networks. Chord [14] is one of the popular structured P2P protocols. Chord is usually deployed on application layer as a P2P overlay. A generic mapping of Chord protocol to Delay-tolerant Network is, however perceived as difficult. We investigate approach that improves the efficiency of the peer/resource lookup algorithm in DTN. The strategies that we use to reduce lookup latency are lowest delay neighbor selection, large routing tables and also recursive lookup routing. With lowest delay neighbor selection, a node chooses each of its neighbors to be the one with the lowest network delay among a set of qualified nodes. The actual network delay of each hop is reduced even though the number of hops remains the same. An alternative is to increase the per-node routing table size, when bandwidth is available. Intuitively, the more neighbors each node knows about, the fewer hops are required during lookups.

3.2 Design Challenges

To find a particular piece of data within the network current DTN applications typically provide lookup functions using controlled-flooding mechanisms. With this approach, the querying node wraps the query in a single message and sends it to all known neighbors.

The neighbors then check to see whether they can reply to the query by matching it to keys in their internal database. If they find a match, they reply; otherwise, they forward the query to their own neighbors. However, flooding-based systems don't scale well because of the bandwidth and processing requirements they place on the network, and they provide no guarantees as to lookup times or content accessibility. Overlay networks can address these issues. Overlay networks have a network semantics layer above the basic transport protocol level that organizes the network topology according to the nodes' content, implementing a distributed hash table abstraction that provides load balancing, query forwarding, and bounded lookup times. Current overlay networks are useful for applications that require reliable, highly scalable, and self organizing storage and lookup for unique key–value pairs. This includes distributed databases, processing clusters, and deterministic search applications. Peer-to-peer overlays can efficiently operate in stable networks where churn rate is low. In highly dynamic networks like DTN, the routing table maintenance cost is very high. DHT's have a routing table comprised of neighbors. In the original Chord DHT proposal, algorithm made this choice of neighbors purely deterministic (*i.e.*, given the set of identifier in the system, the neighbors tables were completely determined). Given a set of neighbors, and a destination, the routing algorithm determines the choice of the next hop. The problem of deterministic neighbor selection is: all links in the "Routing Network" may have high latency.

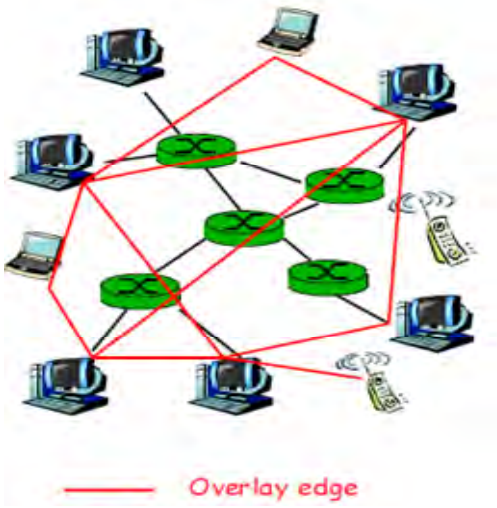


Figure 3.1: Overlay Network

The latency of a lookup is the time taken to route a message to the responsible node for the key and receive a reply back. Low lookup latency is crucial for building fast DHT-based applications over DTN. Most existing work on optimizing DHT performance focuses on achieving low latency in static networks. This latency depends largely on two factors: the average number of hops per lookup (*i.e.* the underlying network delay incurred at each hop) and the average number of timeouts incurred during a lookup. A node can aggressively maintain the freshness of a smaller routing table (thus minimizing timeouts), or to look for new nodes to enlarge the table (thus minimizing lookup hops but perhaps risking timeouts). Highly dynamic network like DTN experiences churn: nodes continuously join and leave the system. Churn poses two problems for routing. First, it causes routing tables to become out of date and to contain stale entries that point to neighbors that are dead or have already left the system. Stale entries result in expensive lookup timeouts as it takes multiple round-trip time for a node to detect a lost lookup message and re-route it through a different neighbor. In static networks, the number of lookup hops and the network delay at each hop determine the end-to-end lookup latency. Under churn, timeouts dominate latency. Second, as new nodes join the system and stale routing entries are deleted, nodes need a way to replenish their routing tables with new entries.

The ideal protocol should be able to adapt its routing table size to provide the best performance using bounded communication overhead. In addition to deciding on the best table size, a DHT should choose the most efficient way of spending bandwidth to keep routing tables up to date under churn. For example, a node could periodically ping each routing entry to check its liveness and search for a replacement entry if an existing neighbor is found to be dead. Intuitively, the faster a node pings, the less likely it is that lookups will encounter timeouts. However, periodic pinging generates overhead messages. The more a node pings, the less bandwidth it has for other uses. In fact, all techniques to cope with churn require extra communication bandwidth in order to evaluate the liveness of existing neighbors and learn about new neighbors. Intuitively, bandwidth consumption increases with the size of the routing table and the churn rate in the network. In other words, churn is a challenge because nodes can only use a finite amount of bandwidth resource. Therefore, the goal of DT-Chord is not to simply achieve low lookup latency

under churn, but to achieve low latency efficiently with bounded bandwidth overhead. In other words, we are interested in the latency reduction per byte of communication overhead, also referred to as latency versus bandwidth tradeoff.

A DHT's routing structure determines from which regions of identifier space a node chooses its neighbors. The ideal routing structure should be both flexible and scalable. With a flexible routing structure, a node is able to expand and contract the size of the routing table along a continuum in response to churn and available bandwidth. With a scalable routing structure, even a very small routing table can lead to efficient lookups in a few hops. However, most DHT routing structures are scalable but not flexible and constrain about routing table sizes are possible. In Chord, the expected number of neighbors per node in a network of n DHT nodes is $(b-1)\log_b n$. The parameter base (b) controls the table size, but it can only take values that are powers of 2, making it difficult to adjust the table size smoothly.

3.3 DT-Chord (Delay Tolerant Chord)

We propose DT-Chord enhancement to improve the performance of base Chord over Delay Tolerant Network. Figure 3.2 shows an illustrative example of a DT-Chord overlay network over DTN and Figure 3.3 shows the DT-Chord application over DTN. The original Chord proposal defines a specific set of neighbors for a given node identifier. Specifically, routing can be achieved in $O(logN)$ hops even if node a were to pick its i^{th} neighbor as any node in the ID space range $a+2^i$ *to* $a+2^{i+1}-1$ rather than the exact node closest to $a+2^i$ as originally defined by Chord [14, 32], where $0 \leq i < m$ (m= no. of bits in the ID space) and base, b=2. The main idea of our scheme is to maintain the lowest delay node for each finger in its ID-space range. To obtain each i^{th} finger, a node retrieves the successor list from a node with ID 2^i away from itself and chooses the node in the ID space range $[a+2^i, a+2^{i+1})$ closest in network latency to itself as the i^{th} finger. While DTN applications are expected to be tolerant of delay, this does not mean that they would not benefit from decreased delay. Minimizing delay lowers the time messages spend in the network, which improves the probability of successful lookup and also reduces the communication cost.

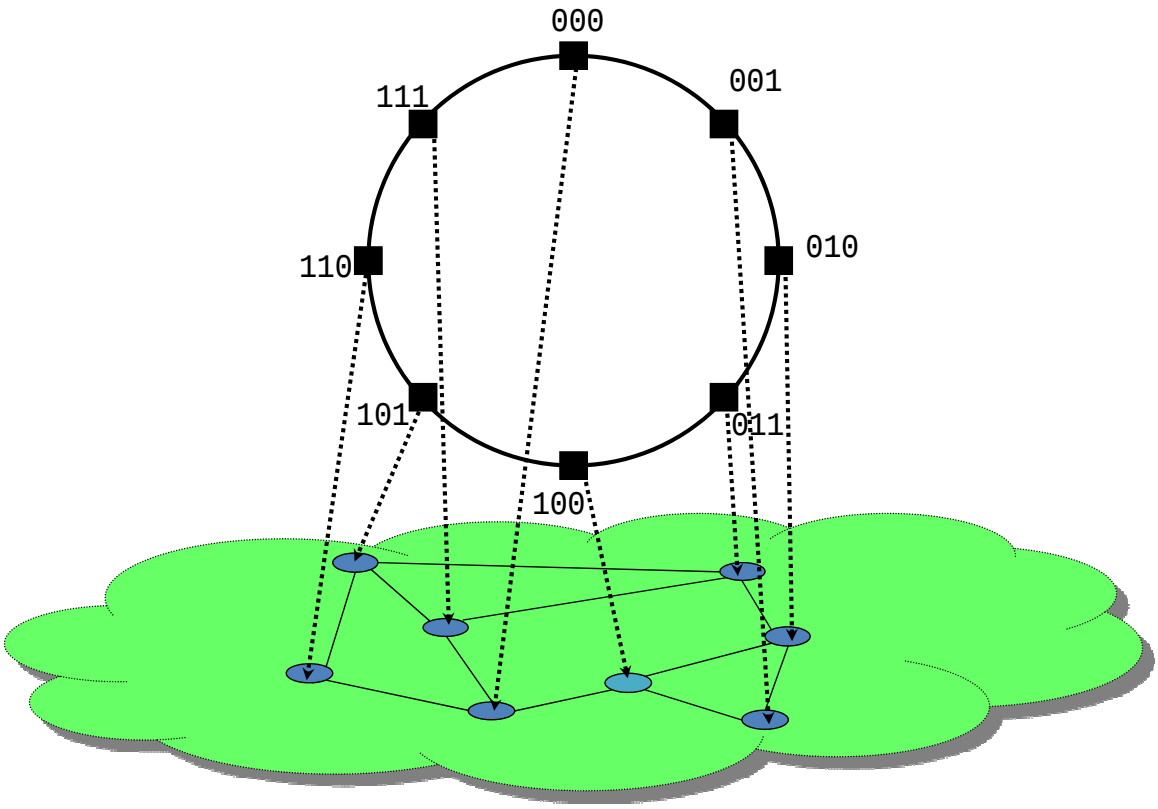


Figure 3.2: An example of DT-Chord overlay over DTN

An optimization to Chord is increasing the amount of information that Chord keeps about other nodes in the system. One way to do this would be to change the base of the finger table. By keeping a larger finger table, each hop could move ¾ of the way to the target, for example, instead of half way. In general, by increasing the size of the routing table to $(b\text{-}1)\log_b N$, Chord can achieve $\log_b N$ hop lookups [14]. These optimizations would reduce latency under low churn, because each node would know about many other nodes. On the other hand, in high churn networks, these optimizations require more bandwidth to keep routing tables up to date and experience more timeouts because routing tables contain recently-failed nodes.

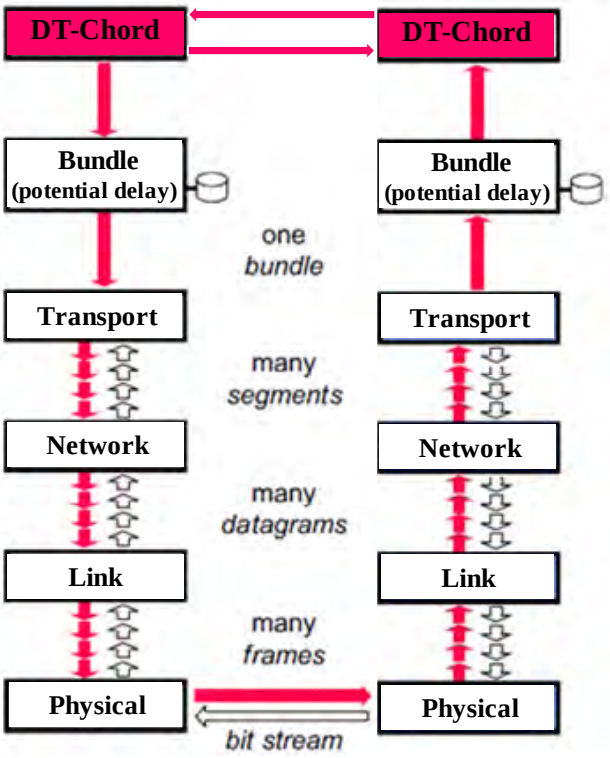
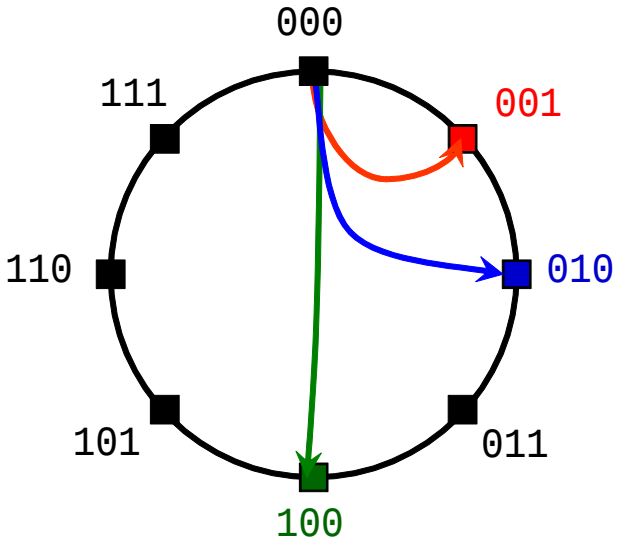


Figure 3.3: DT-Chord Application over DTN Protocol Stack

3.3.1 DT-Chord Neighbor Selection

In DT-Chord, a node in base-b keeps $(b-1)\log_b(n)$ fingers whose IDs lie at exponentially increasing fractions of the ID space away from itself. Any node whose ID lies within the range $x+\left(\frac{b-1}{b}\right)^{i+1}*2^m$ and $x+\left(\frac{b-1}{b}\right)^{i}*2^m$, modulo 2^m, can be used as the i^{th} finger of x, where m is the number of bits in the ID space and $0 \le i < m$. To obtain each i^{th} finger, a node retrieves the successor list of n_{succ} nodes from a node with ID $x+\left(\frac{b-1}{b}\right)^{i}$ away from itself and chooses the node closest in network latency to itself as the i^{th} finger. Each node also keeps a *successor list* of size n_{succ}, containing the node's first n_{succ} successors. A node x periodically pings all its fingers to check their liveness. For each finger found dead, the node issues a lookup for replacement of finger. A node separately stabilizes its successor list by periodically retrieving and merging its successor's successor list; successor stabilization is separate because it is critical for correctness but is much cheaper than finger stabilization. Figure 3.4 and 3.5 show a simple scenario of Chord and DT-Chord

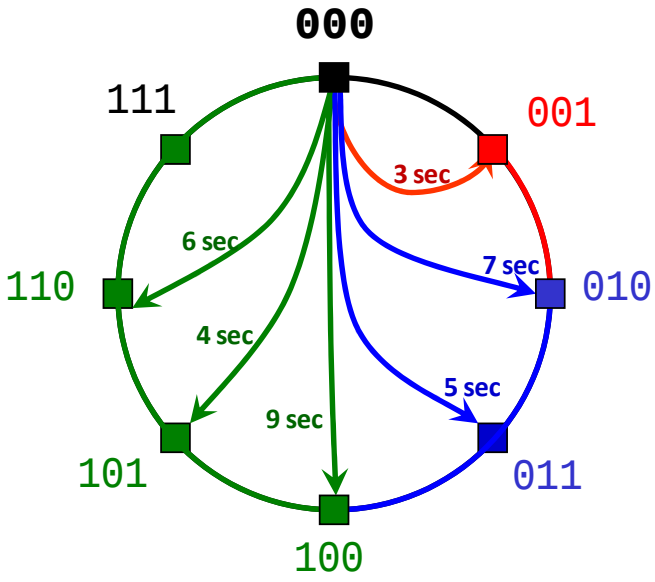
neighbor selection and Figure 3.6 and 3.7 show the Chord and DT-Chord neighbor selection example in detail.



Chord Finger Table of 000

Key	Value
000 + 001	001
000 + 010	010
000 + 100	100

Figure 3.4: Base Chord has a deterministic and rigid finger table



Successor List

Node	Successor List
000	001, 010
001	**010,011**
010	**011, 100**
011	100, 101
100	**101, 110**
---	--- ---

DT-Chord Finger Table of 000

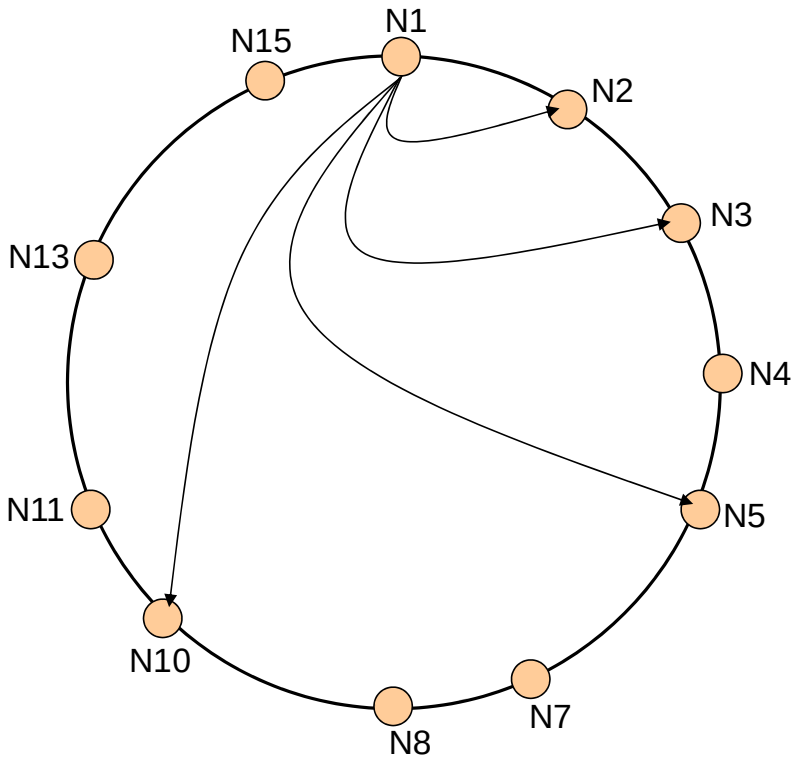
	Range	Choose
000 + 001	[001, 010)	001
000 + 010	[010, 100)	011
000 + 100	[100, 000)	101

Figure 3.5: DT-Chord's finger table. In the figure, to obtain 1st, 2nd and 3rd finger, DT-Chord node 000 retrieves the successor list of node 001, 010, 100 and chooses the node in the ID space range [001, 010), [010, 100), [100, 000) respectively closest in network latency to itself (Here, 001, 011 and 101). In this way, it will choose every node's finger's entry.

3.3.2 Chord and DT-Chord Neighbor Selection Example in Detail

(a) Chord Neighbor Selection

Let the number of bits in the key/node identifiers are 4 and *base*=2. Each node *n* maintains a routing table with up to 4 entries, called the finger table. The i^{th} entry in the table at node *n* contains the identity of the first node *s* that succeeds *n* by at least 2^i on the identifier circle, *i.e.*, $s = successor(n+2^i)$, where $0 \leq i < m$ (and all arithmetic is modulo 2^m). Note that the first finger of *n* is the immediate successor of *n* on the circle; for convenience we often refer to the first finger as the successor. To increase robustness, each Chord node maintains a successor list of size *r*, containing the node's first *r* successors.



Chord Finger Table of Node 1 (N1)

$N1+2^0$	N2
$N1+2^1$	N3
$N1+2^2$	N5
$N1+2^3$	N10

Figure 3.6: Chord finger table entries for node N1

The example in figure 3.6 shows the base Chord finger table of node 1 (N1). The first finger of node 1 points to node 2, as node 2 is the first node that succeeds $(1+2^0) \bmod 2^4 = 2$. Similarly, the last finger of node 1 points to node 10, as node 10 is the first node that succeeds $(1+2^3) \bmod 2^4 = 10$.

(b) DT-Chord Neighbor Selection

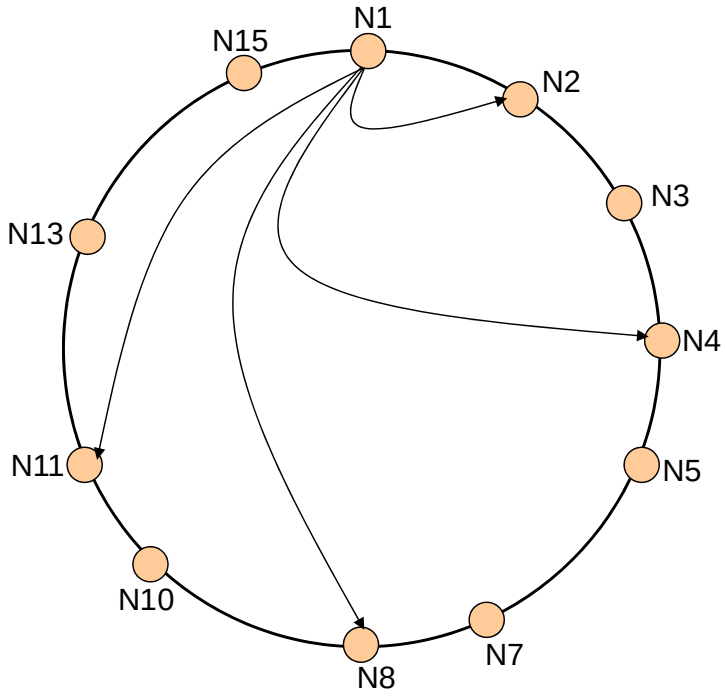
Let, each DT-Chord node have 2 successors. The successor list of every nodes and the pair wise latency between node N1 and other nodes in the systems are given below:

Node	Successor List
1	2, 3
2	3, 4
3	4, 5
4	5, 9
5	7, 8
7	8, 10
8	10, 11
10	11, 13
---	--- ---

Table 3.1: Successor list of Chord nodes

Node	Delay
2	4 sec
3	5 sec
4	3 sec
5	7 sec
7	9 sec
8	4 sec
10	6 sec
11	5 sec
13	8 sec
---	--- ---

Table 3.2: Pair wise latency of node N1



	Range	Choose
$N1+2^0$	[2, 3)	N2
$N1+2^1$	[3, 5)	N4
$N1+2^2$	[5, 9)	N8
$N1+2^3$	[9, 16)	N11

Figure 3.7: DT-Chord finger table entries for node N1

The example in Figure 3.7 shows the DT-Chord finger table of node N1. The *1st finger* of node N1 points to node 2, as node 2 is the only node in the range [2, 3). So, there is no option to choose lowest delay neighbor. To obtain *2nd finger*, node N1 retrieves the successor list from the node N3 ($N1+2^1$= N3) and chooses the node from the list [3, 4] that lies in the ID space range *[3, 5)* closest in network latency to itself. From the table 3.2, we can see that the 2nd finger of N1 will be N4. Similarly to obtain *3rd finger*, node N1 retrieves the successor list from the node N5 ($N1+2^2$= N5) and chooses the node from the list [5, 7, 8] that lies in the ID space range *[5, 9)* closest in network latency to N1. So, the 3rd finger of N1 will be N8. The *last (4th) finger* of N1 will be N11, as N11 is the node that has lowest latency to N1, among the node N10 ($N1+2^3$= N10, as there is no node, N9) and its successor list that lies in the ID space range *[9, 16)*.

3.3.3 DT-Chord Route Selection

The base Chord lookup routing algorithm is iterative. We use recursive lookup routing algorithm in DT-Chord. Here, the requesting node forwards the lookup to the first hop which forwards in turns forwards the request to the next best hop (instead of returning information about the next best hop to the requester). When the lookup reaches the key's predecessor, the predecessor sends a message to the node that originated the lookup with the results of the lookup. Figure 3.8 shows the pseudo code for recursive lookup using finger tables. Note that no messages are sent to the originating node prior to the lookup's completion.

While recursive lookup has lower latency than iterative, iterative is easier for a client to manage. If a recursive lookup elicits no response, the originator has no information about what went wrong and how to re-try in a way that is more likely to succeed. Sometimes a simple re-try may work, as in the case of lost packets. If the problem is that each successive node can talk to the next node, but that Internet routing anomalies prevent the last node from replying to the originator, then re-tries won't work because only the originator realizes a problem exists. In contrast, the originator knows which hop of an iterative lookup failed to respond, and can re-try that hop through a different node in the

same region of the identifier space. On the other hand, recursive communication may make congestion control easier. DT-Chord uses recursive lookups by default since they are faster, but could fall back on iterative lookups after persistent failures.

```
// ask node n to find the successor of id
// This lookup is being done on behalf of node orig
n.find_successor(id, orig)
        if (id ∈ (n, successor])
                orig.lookup_done(successor_list);
        else
        n´ = closest preceding_node(id);
        n´.find_successor(id);

// search the local table for the highest predecessor of id
n.closest_preceding_node(id)
        for i = m downto 1
                if (finger[i] ∈ (n, id))
                        return finger[i];
        return n;
// called when a lookup completes. Return the results
// of the lookup to the user
n.lookup_done(successors)
```

Figure 3.8: DT-Chord's recursive lookup using the finger table. **find_successor** returns the successors of key *id* by forwarding the lookup to the finger table entry that is closest to the target. Note that *finger*[0] is the node's successor.

3.3.4 Routing State Freshness

A node must strike a balance between the freshness and the size of its routing table. Nodes need to judge the freshness of entries to decide when to evict nodes, in order to limit the number of expected lookup timeouts. Timeouts are expensive as nodes need to wait multiple round trip times to declare the lookup message failed before re-issuing it to a different neighbor. In order to avoid timeouts, DT-Chord nodes contact each neighbor periodically to determine the routing entry's liveness. In other words, a node can control its routing state freshness by evicting neighbors from its routing table that it has not successfully contacted for some interval. If the available bandwidth were infinite, the node could ping each neighbor often to maintain fresh tables of arbitrarily large size. However, with a finite bandwidth, a node must somehow make a tradeoff between the freshness and the size of its routing table.

Since nodes need to keep their maintenance traffic according to available bandwidth, they can only refresh or learn about new neighbors at some finite rate. For example, if a node's available bandwidth is 20 bytes per second, and learning liveness information for a single neighbor costs 4 bytes (*e.g.*, the neighbor's IP address), then at most a node could refresh or learn routing table entries for 5 nodes per second.

3.4 Summary

In this chapter, we have proposed a DHT based lookup protocol for DTN, DT-Chord - an efficient lookup protocol that minimizes delay while locating data in DTN and have discussed the theoretical background of DT-Chord.

Chapter 4

Evaluation

4.1 P2P Simulators

In this section we enlisted some P2P simulators. The P2P research community has not come to a consensus on standardizing a simulation platform for simulating the research projects from numerous working groups. From the survey presented in [24], it can be observed that more than 90% of P2P research was tested in non-standard or custom-made simulation environments. Yet there are a number of freely available P2P-simulators on the Internet. Most of these simulators are still at the early stages of implementation; it will take a while for these simulators to achieve maturity. A comprehensive survey on P2P simulators can be found in [24].

For our experiments we have used P2PSim, which is described in the next section. Few P2P simulators that we considered as possible alternatives for P2PSim are listed below:

- PlanetSim [25]
- PeerSim[26]
- GPS [27]
- 3LS [28]
- Query-Cycle Simulator [29]
- NeuroGrid [33]
- PlanetSim [34]

4.2 P2PSim

We have implemented DT-Chord in P2PSim [23] for our experiments. P2PSim is an open source, discrete event simulator intended to compare, evaluate, and explore peer-to-peer protocols. Like other P2P simulators it does not consider the underlying network communication stack for monitoring network layer performance, rather the focus is on the overlay layer. In our simulation, the simulated network models only packet delay. One input to the simulator is a full matrix of the round-trip delays between each pair of simulated hosts. This approach avoids having to simulate the Internet's topology, a currently open area of research; it requires only the measurement of actual pair-wise delays among a set of hosts. The simulator can produce useful speed-of-light delay results, but cannot be used to predict throughput or queuing delay. This suffices for our experiments where network latency is the bottleneck.

4.3 Experimental Dataset

The simulated network, consists of 128, 256, 512, 1024 nodes with a pair wise latency matrix derived from measuring the inter-node latencies of 128, 256, 512, 1024 DNS servers respectively using the King method [30].

4.4 Simulation Design

In this section, we describe the main components of our simulation, explains how they communicate, and discusses simulator's control flow design.

4.4.1 Main components

A simulation includes Nodes, a Network, a Topology, a Lookup Generator, and a Churn Generator. Figure 4.1 gives an overview of these components, and how they interact.

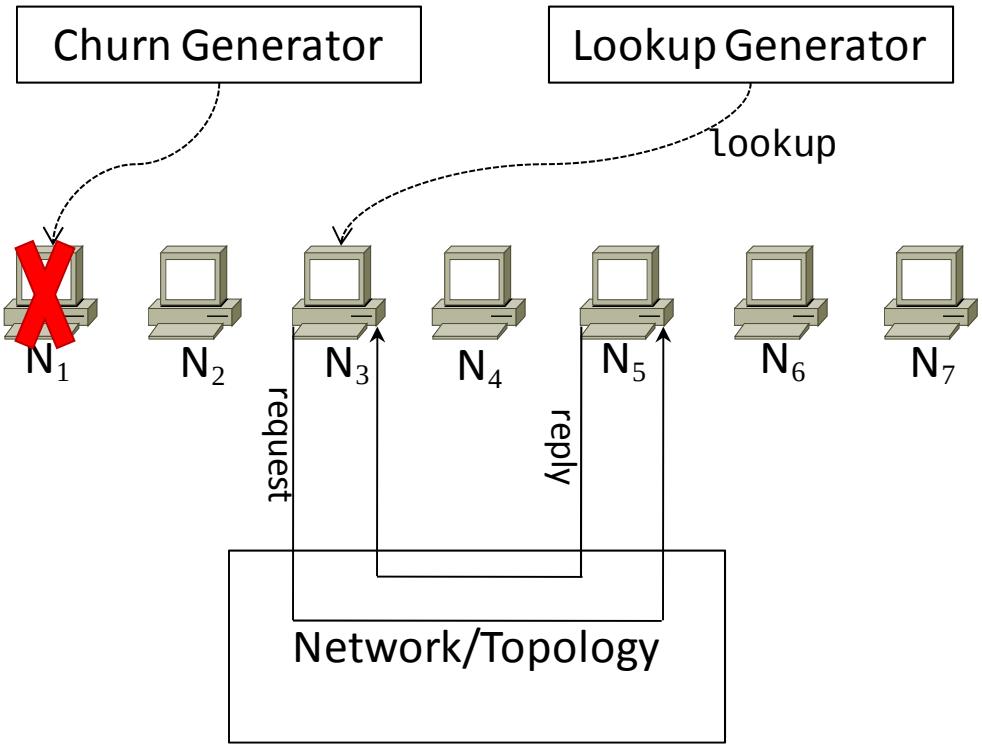


Figure 4.1: Major components of simulation

A Node simulates a computer running a peer-to-peer algorithm. Nodes (labeled N1 to N7 in Figure 4.1) communicate to each other by sending packets through the Network. The Network uses the Topology to determine the latency between two nodes. The Churn Generator models the dynamic nature of delay tolerant networks. Periodically, it either adds a node to the network by invoking the node's join method, or removes a node from the network by destroying it. Figure 4.1 shows the Churn Generator removing node N1. The Lookup Generator simulates a workload by periodically issuing lookup requests. Figure 4.1 shows the Lookup Generator generating a lookup request for node N3. To satisfy this request, node N3 sends a packet over the Network to Node N5, N5 sends back a reply, and N3 logs the time it took for the lookup to complete. A node doing a lookup runs the protocol's lookup algorithm to find node N closest to a given key K in identifier space. Periodically, a Node may run a stabilization routine to refresh the entries in its routing table.

Nodes communicate to each other by issuing RPCs that are sent in packets through the Network. Upon receiving a packet, the Network asks the Topology for the latency

between the sender and receiver, which determines the correct simulated time at which the packet is delivered to the destination Node. When an RPC is issued to a dead node, the Network mimics a timeout by scheduling a special error-RPC-reply *n* roundtrip times after the RPC was sent, where *n* is some user-configurable value.

4.5 Evaluation Criteria

Lookup performance has often been measured with hopcount, latency, success rate, and probability of timeouts. Lookup latency alone is not sufficient to evaluate protocol under churn, where nodes continuously join and leave the network, because the latency metric does not account for the cost of maintaining the state required to achieve low latency. Evaluating lookup performance in static networks tends to favor protocols that keep large routing tables, since they pay no penalty to keep the tables' up to date, and more routing entries generally results in lower lookup hop-counts and latencies. Large routing tables incur costs, however they require maintenance traffic to keep them up to date, and if they become out of date then stale entries may cause timeout delays. Thus an evaluation criterion for protocol under churn should reflect the relationship between latency and cost.

4.6 Comparison Framework

Two challenges exist in evaluating DT-Chord lookup protocol over DTN. First, protocol can be tuned to have low lookup latency by including features such as aggressive membership maintenance, faster routing table liveness checking, or a more thorough exploration of the network to find low delay neighbors. Any evaluation that examines how DT-Chord performs along one dimension of either cost (in terms of bandwidth consumed) or performance (in terms of lookup latency) is flawed, since an analysis can "cheat" by performing extremely well on the axis being measured but terribly on the other. Thus a comparison of lookup protocol must consider the performance and cost simultaneously, *i.e.* the efficiency with which it exploits bandwidth to reduce latency. The efficiency can be characterized by a performance vs. cost tradeoff curve: at any given bandwidth, there is one best achievable latency. However, efficiency cannot be measured

by a single number summarizing the ratio between a protocol's bandwidth consumption and its lookup latency, as the tradeoffs between performance and cost does not necessary follow a linear relationship.

The second challenge is to cope with protocol's set of tunable parameters (*e.g.*, liveness checking interval, routing table size *etc.*). The best parameter values for a given workload are often hard to predict, so there is a danger that a performance evaluation might reflect the evaluator's parameter choices more than it reflects the underlying algorithm. In addition, parameters often correspond to a given protocol feature. A good analysis should allow designers to judge the extent to which each parameter (and thus each feature) contributes to overall bandwidth efficiency.

In response to these two challenges, we propose a comparison framework and evaluation methodology for assessing DT-Chord protocol, comparing different design choices and evaluating new features.

4.6.1 Performance Metrics

We measure performance as the lookup failure rate and the average lookup latency of correct lookups (*i.e.*, lookups for which a correct answer is returned), including timeout penalties (three times the round-trip time to the dead node). Protocols retry failed lookups (*i.e.*, lookups that time out without completing) for up to a maximum of four seconds. We only incorporate lookup hopcount indirectly, to the extent that it contributes to latency. In the presence of churn, routing tables tend to become incorrect or out of date, causing lookups to suffer timeouts or completely fail.

4.6.2 Cost Metric

We measure cost as the average bandwidth consumed per node per alive second (*i.e.*, we divide the total bytes consumed by the sum of times that each node was alive). The size of each message is counted as 20 bytes for headers plus 4 bytes for each node mentioned in the message. This cost accounts for all messages sent by a node, including periodic

routing table refresh traffic, lookup traffic, and join traffic. We ignore state storage costs (*e.g.*, the size of each node's routing table) because communication is typically far more expensive than storage (memory) or CPU time. The main cost of state is often the communication cost necessary for maintaining the correctness of that state.

4.7 Experimental Environment

In our simulation, nodes try to forward lookups to the node responsible for the lookup key. The identity of the responsible node is returned to the sender as the result of the lookup. A lookup is considered failed if it returns the wrong node among the current set of participating nodes (*i.e.* those that have completed the join procedure correctly) at the time the sender receives the lookup reply, or if the sender receives no reply within some timeout window. The evaluation framework accounts for the cost of trying to contact a dead node during a lookup as a latency penalty equal to a small constant multiple of the round trip time to the dead node, an optimistic simulation of the cost of a timeout before the node pursues the lookup through an alternate route. In our experiments, protocol time out individual messages after an interval of three times the round-trip time to the target node. A node encountering a timeout to a particular neighbor during a lookup does not immediately declare that neighbor dead; the lookup proceeds to an alternate node if one exists, and recovery does not begin for the failed neighbor until 5 RPC timeouts to that neighbor occur. Protocol retries alternate routes for lookups for up to a maximum of four seconds, after which the lookup has declared fail. This definition of failure is arbitrary: a shorter maximum time would decrease average latency while increasing failure rate, while a longer maximum would increase average latency while decreasing the failure rate. Further, each failed lookup contributes a disproportionate four seconds to the average lookup latency statistic. For these reasons, we measures lookup failure rate and average lookup latency as separate performance metrics.

The average roundtrip delay between node pairs in our dataset is 156 ms. Since each lookup for a random key must terminate at a specific, random node in the network, the average latency of the topology serves as a lower bound for the average DHT lookup latency. Each node alternately crashes and re-joins the network; the interval between

successive events for each node is exponentially distributed with a mean of one hour [31]. The amount a protocol must communicate to keep node routing tables up to date depends on how frequently nodes join and crash (the churn rate). For the most part, the total bandwidth consumed by a protocol is a balance between table maintenance traffic and lookup traffic, so the main characteristic of a workload is the relationship between lookup rate and churn rate. We investigate two workloads, one that is churn intensive and one that is lookup intensive. In the churn intensive workload, each node issues lookups for random keys at intervals exponentially distributed with a mean of 600 seconds. In the lookup intensive workload, the lookup interval mean is 9 seconds. Unless otherwise noted, all figures are for simulations done in the churn intensive workload. Each simulation runs for six hours of simulated time; statistics are collected only during the second half of the simulation and averaged over 5 simulation runs.

4.7.1 Simulation Parameters

Table 4.1 lists the Chord and DT-Chord parameters that we vary in our simulations.

	Parameter	**Range**
Number of Successors	n_{succ}	8 ,16
Successor Stabilization Interval	t_{succ}	9 sec – 19 min
Amount of state (Finger Base)	b	2, 4, 8, 16, 32
Freshness of State (Finger Stabilization Interval)	t_{finger}	9 sec – 19 min

Table 4.1: Chord and DT-Chord Simulation Parameters

4.7.2 Results

We systematically simulate DT-Chord with different combinations of parameter values. For each parameter combination, we plot the performance and cost measured from the experiment on a graph with total bandwidth usage on the *x*-axis and average lookup latency in milliseconds or failure rate on the *y*-axis. For example, in Figure 4.2, each of the many hundred points corresponds to a different parameter combination under a particular workload. A point that lies to the lower left of another point is more efficient as its corresponding parameter combination results in both lower lookup latency and lower bandwidth consumption.

To characterize the efficiency of DT-Chord, we need to find the best set of performance vs. cost tradeoff points that correspond to the optimal parameter settings. As can be seen in Figure 4.2, there is no single best performance vs. cost tradeoff point. Instead, there is a set of best points: for each cost, there is a smallest achievable lookup latency, and for each lookup latency, there is a smallest achievable communication cost. The curve connecting these best points is the overall convex hull segment (shown by the solid line in Figure) that lies beneath and to the left of all points. A convex hull outlines the best achievable performance vs. cost tradeoffs with the optimal parameter settings. A convex hull segment always goes up to the left of the graph as bandwidth decreases. This means that there is no parameter combination that simultaneously produces both low lookup latency (or low failure rate) and low bandwidth consumption. The convex hulls go down at higher bandwidth because there are parameter values that improve lookup latency (or failure rate) at the cost of increased bandwidth consumption.

The convex hull in Figure 4.2 is only for a specific workload and churn scenario being simulated. The best parameter values (thus the overall convex hulls) might change as workloads or the churn rates change. Therefore, the convex hull only outlines maximal efficiency in theory. An operator would have to adjust the protocol parameters manually under known workload and churn scenario in the absence of a self tuning protocol.

In Figures 4.2, 4.4, 4.6, 4.8, each point represents the average lookup latency of successful lookups vs. the communication cost achieved for a unique set of parameter values. In Figures 4.3, 4.5, 4.7, 4.9 each point represents the lookup failure rate vs. the communication cost

achieved for a unique set of parameter values. The convex hull (solid line) represents the best achievable performance/cost combinations.

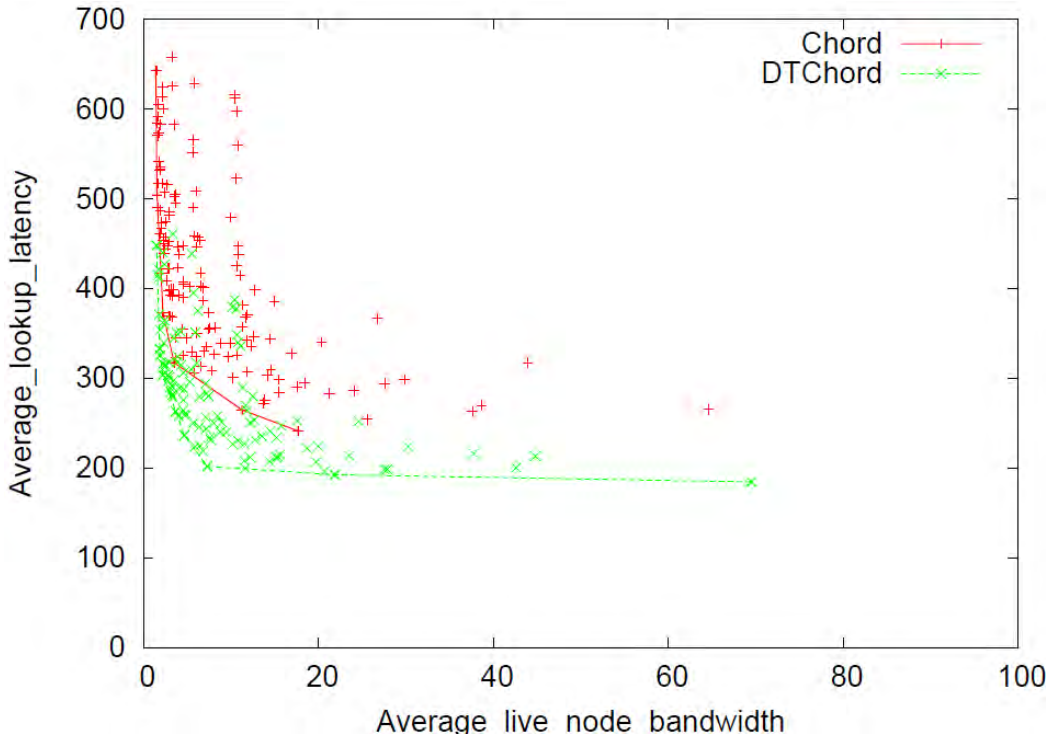


Figure 4.2: Overall convex hulls for successful lookup latency vs. bandwidth tradeoff of Chord and DTChord with network size 128, under the churn intensive workload.

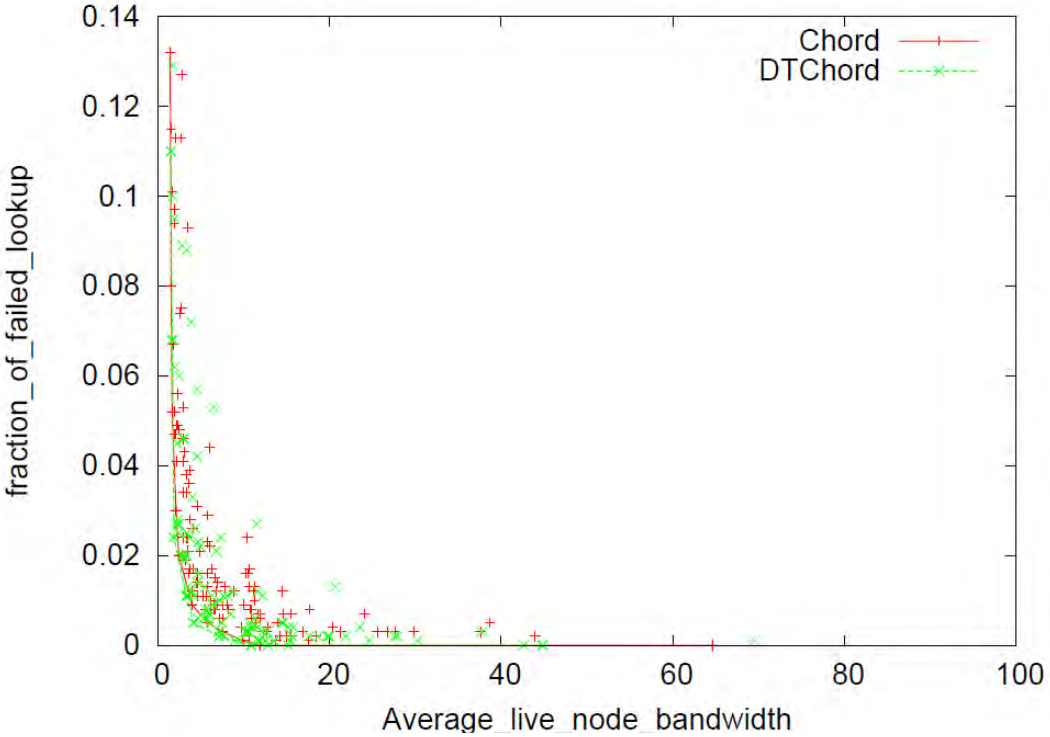


Figure 4.3: Overall convex hulls for lookup failure rate vs. bandwidth tradeoff of Chord and DTChord with network size 128, under the churn intensive workload.

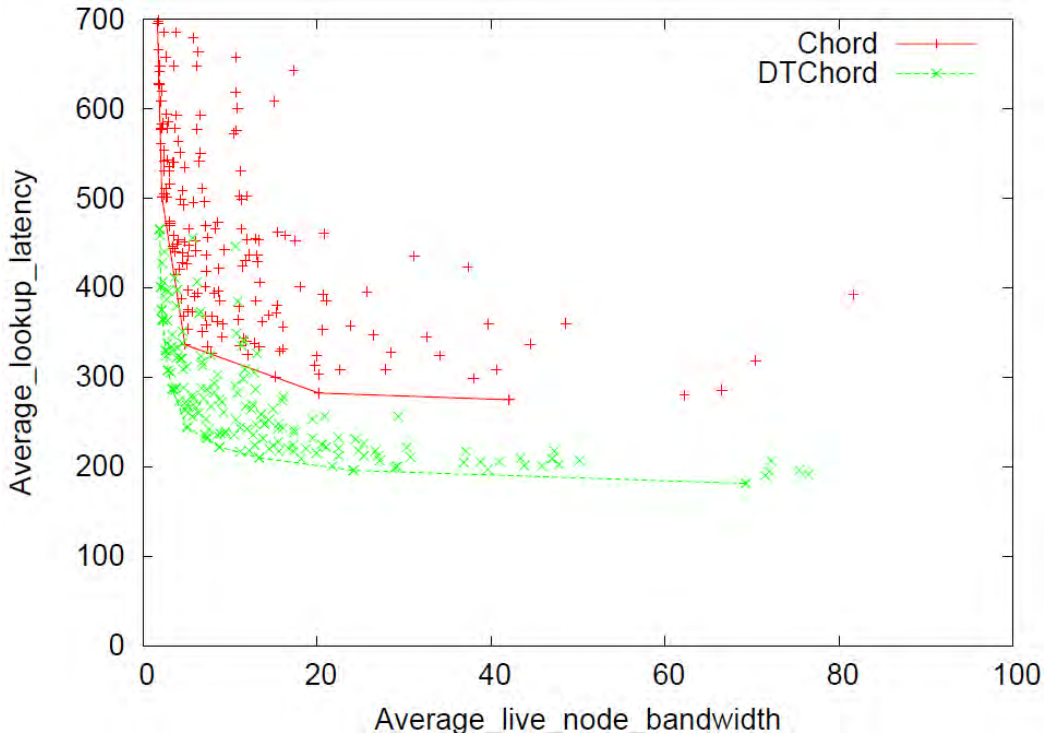


Figure 4.4: Overall convex hulls for successful lookup latency vs. bandwidth tradeoff of Chord and DTChord with network size 256, under the churn intensive workload.

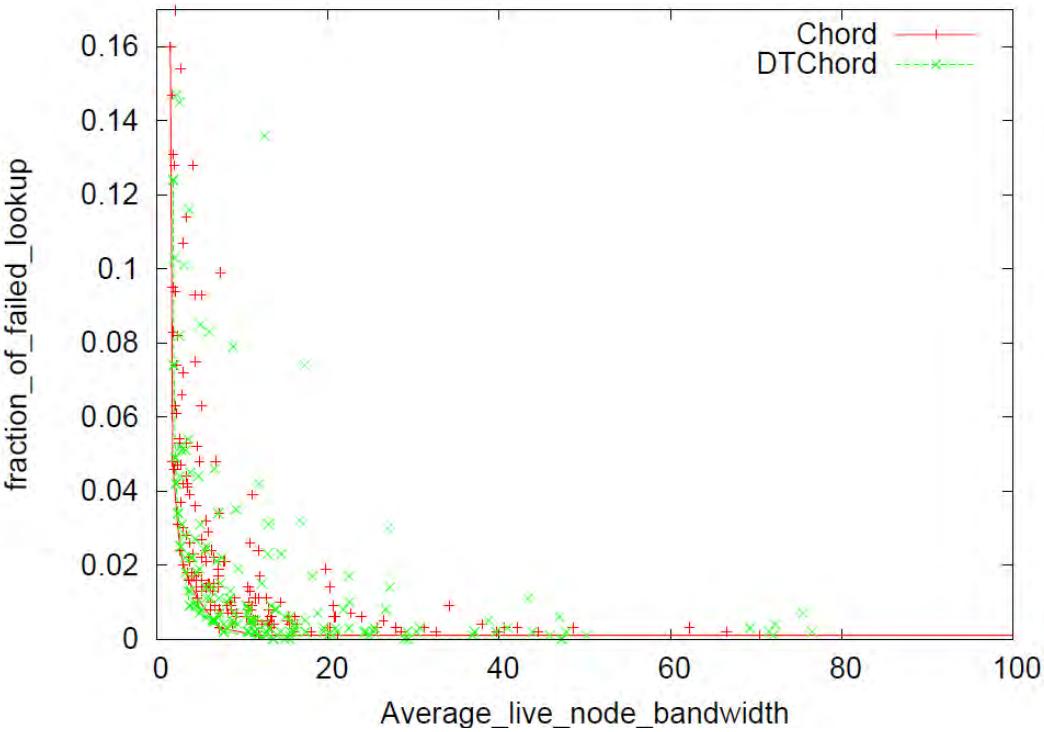


Figure 4.5: Overall convex hulls for lookup failure rate vs. bandwidth tradeoff of Chord and DTChord with network size 256, under the churn intensive workload.

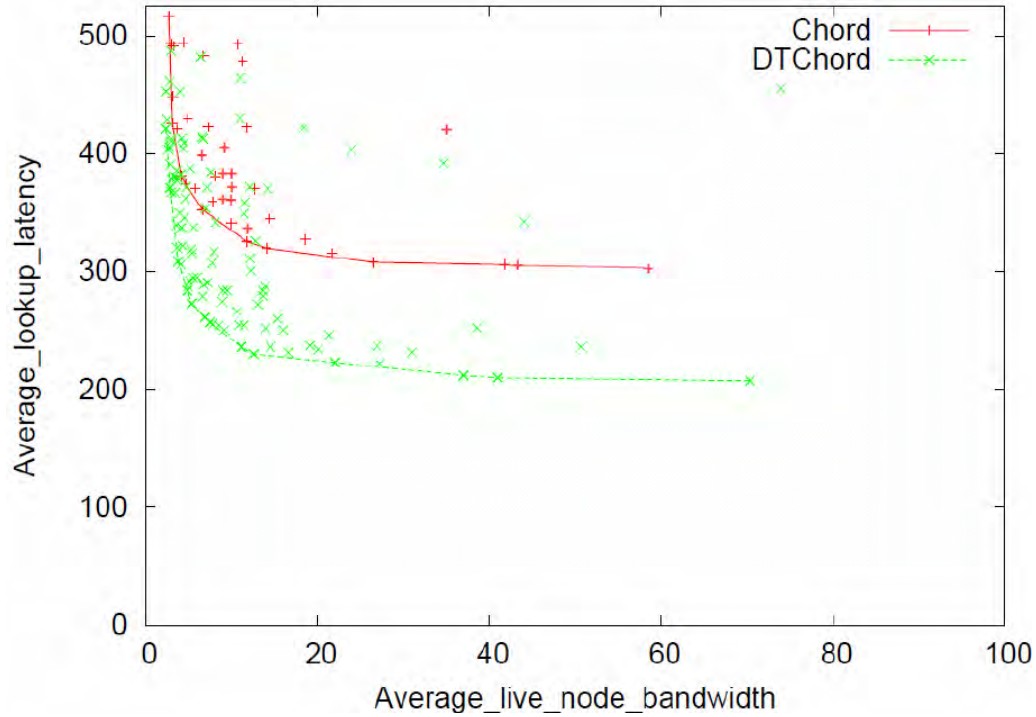


Figure 4.6: Overall convex hulls for successful lookup latency vs. bandwidth tradeoff of Chord and DTChord with network size 512, under the churn intensive workload.

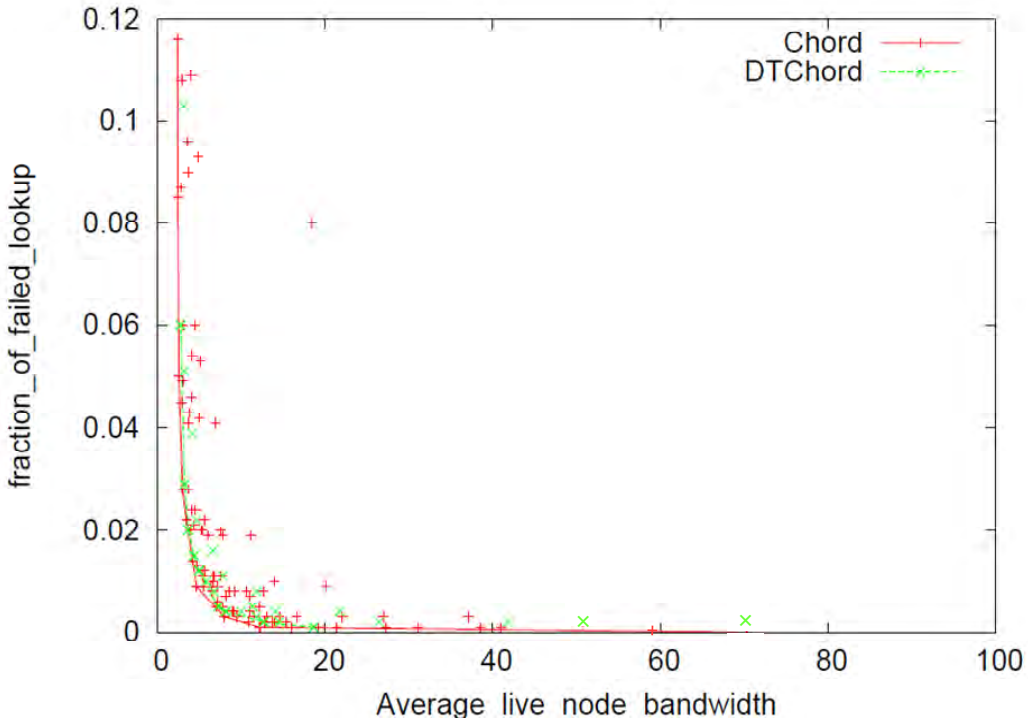


Figure 4.7: Overall convex hulls for lookup failure rate vs. bandwidth tradeoff of Chord and DTChord with network size 512, under the churn intensive workload.

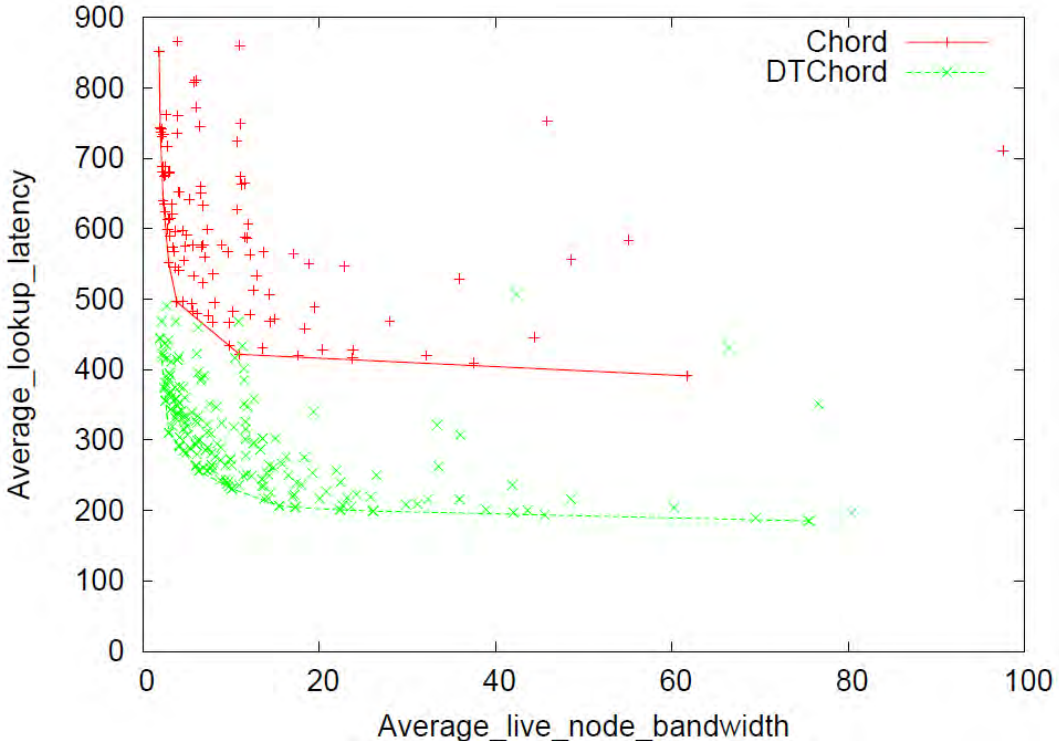


Figure 4.8: Overall convex hulls for successful lookup latency vs. bandwidth tradeoff of Chord and DTChord with network size 1024, under the churn intensive workload.

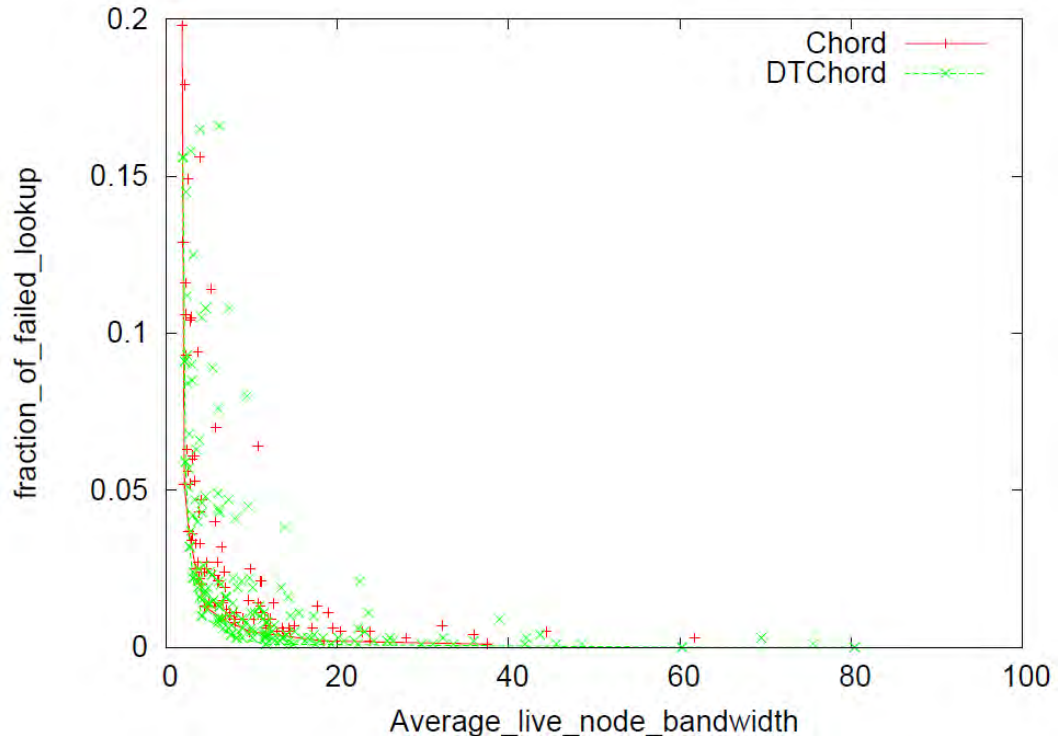
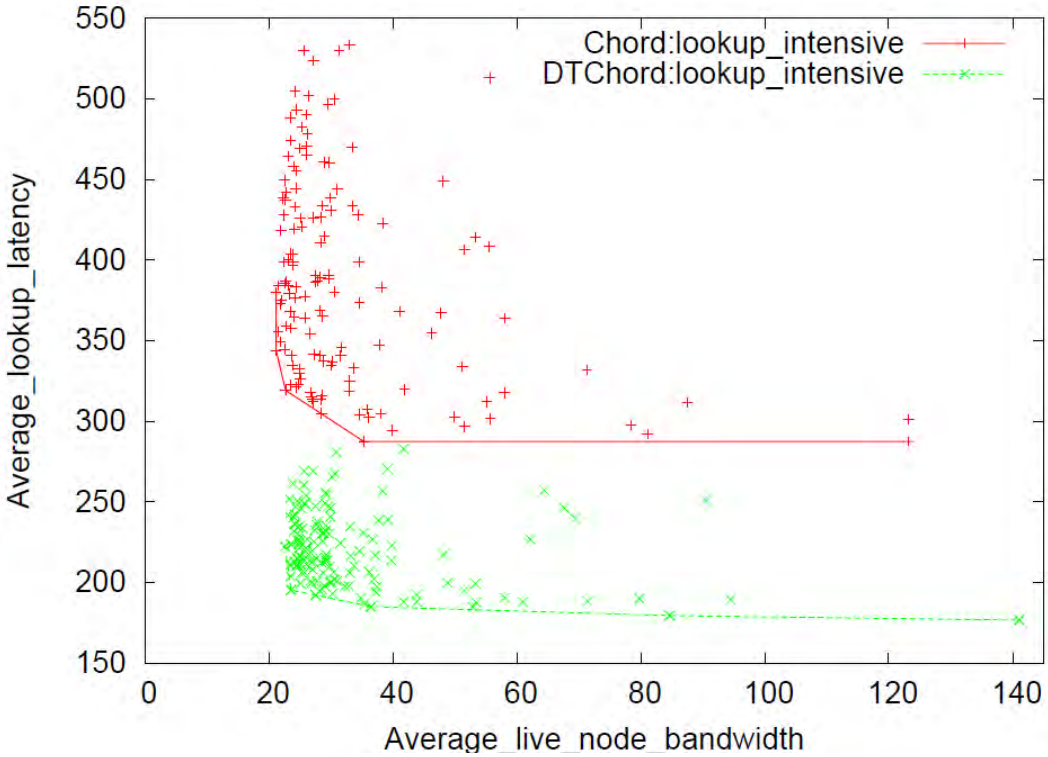


Figure 4.9: Overall convex hulls for lookup failure rate vs. bandwidth tradeoff of Chord and DTChord with network size 1024, under the churn intensive workload.

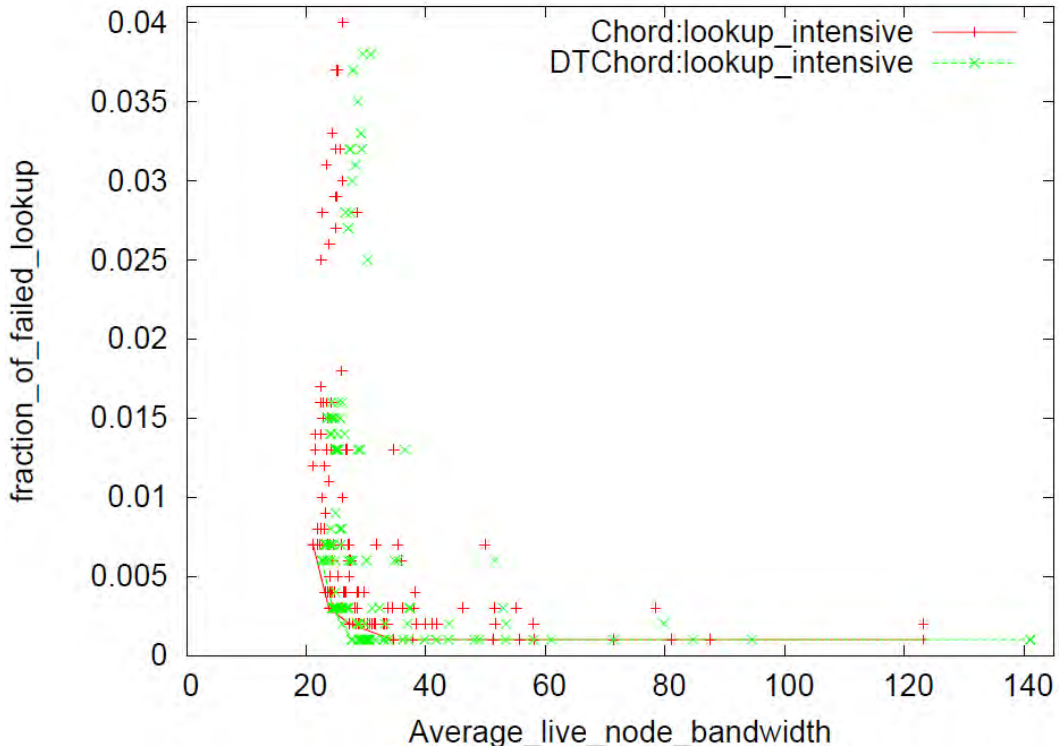
4.8 Effect of Lookup-intensive Workload

The lookup intensive workload involves each node issuing a lookup request every 9 seconds, almost 67 times the rate of the churn intensive workload used in the preceding sections. As a result, the lookup traffic dominates the total bandwidth consumption. Figure 4.10 shows the overall convex hulls of Chord and DT-Chord protocol under the lookup intensive workload with network size 1024.

In the lookup intensive workload, each node issues and forwards much more lookup messages during its lifetime and hence the amount of churn is relatively low. Thus, it is more efficient to keep a larger routing table for fewer lookup hops when the amount of stabilization traffic is low compared to the amount of lookup traffic. Furthermore, fewer lookup hops translate into a large decrease in forwarded lookup traffic, given the large number of lookups.



(a)



(b)

Figure 4.10: Chord and DT-Chord under lookup intensive workload. (a) Each point represents the successful lookup latency vs. the communication cost. (b) Each point represents lookup failure rate vs. the communication cost achieved for a unique set of parameter values.

4.9 Summary

We ran simulations of each Chord and DT-Chord protocol with same combinations of the parameters. The results of the simulations show that proposed DT-Chord outperforms Chord in terms of reducing latency, correct query lookup in different network size and different work load. Both failure rate and lookup latency decrease as the protocols consume more bandwidth.

Chapter 5

Performance Tuning of DT-Chord

In order to design DT-Chord with best lookup performance, we need to understand how to use available bandwidth most efficiently. The efficiency of a protocol measures its ability to turn each extra byte of maintenance communication into reduced lookup latency. In the figures of previous chapter, we have seen the combined effect of many parameters. In these figures, some parameter settings are much more efficient than others. So, it is needed to evaluate whether a particular parameter is more important to tune than others in order to achieve the best performance/cost tradeoff. This chapter identifies the importance of different parameters and relating them to the different design choices. We provide an extensive simulation study to evaluate how efficiently different design choices use additional bandwidth for better lookup performance. This is done by calculating a set of parameter convex hulls, one for each value of the parameter under study. Each parameter convex hull is generated by fixing the parameter of interest and varying all others. Each parameter hull represents the best possible performance vs. cost tradeoffs for a fixed parameter value. In this chapter, the simulated network, unless otherwise noted, consists of 1024 nodes.

5.1 DT-Chord Parameter Analysis

The various bandwidth consumptions and lookup performance are the indirect consequence of setting different parameters to different values. That is, the convex hulls are the result of an exhaustive search for the best parameter values. What parameter values produced the best tradeoffs that make up the convex hull? More importantly, if the available bandwidth changes, what are the parameters that need to be re-adjusted to optimize lookup performance?

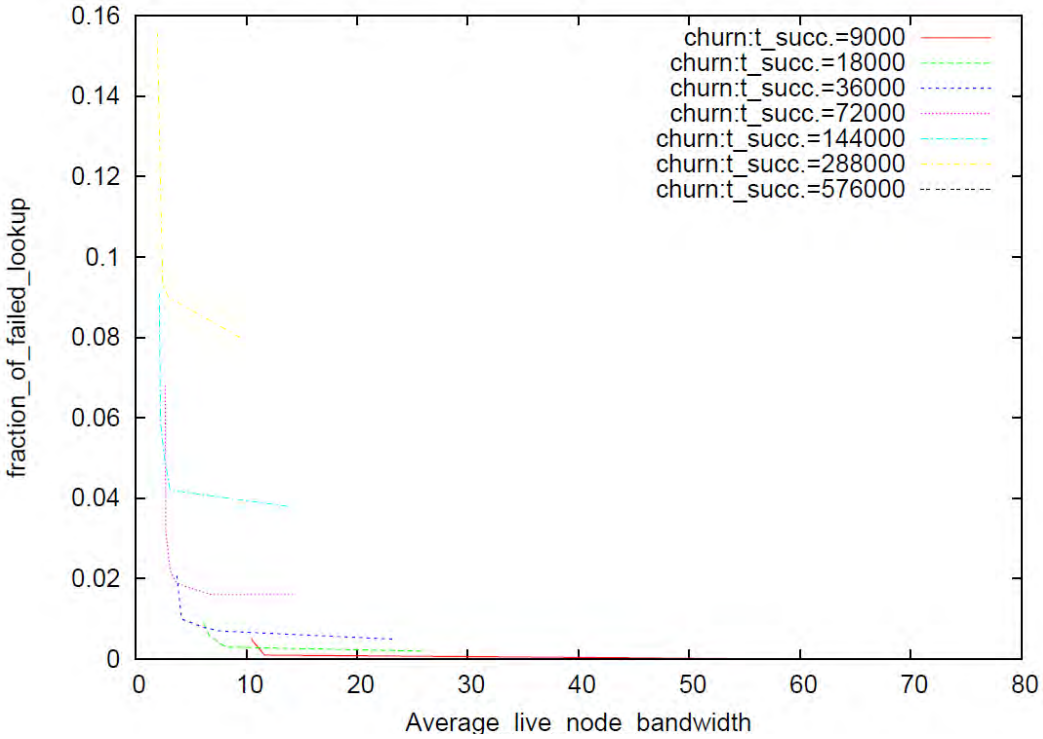
Each parameter corresponds to a design choice. Different design choices compete with each other in using extra bandwidth to improve lookup performance. For example, setting a bigger base (b) or a smaller stabilization interval (t_{finger}) can both lower lookup latency at the cost of increased bandwidth consumption. Therefore, measuring the performance benefits by adjusting a single parameter in isolation can be misleading as it ignores other competitive choices of using bandwidth. We solve this problem with parameter convex hull analysis. Instead of measuring the performance benefits of adjusting the parameter of interest, we examine the efficiency loss from not adjusting the parameter under study and exploring all other parameters. A parameter convex hull outlines the bandwidth efficiency achieved under a fixed value for the parameter under study while exploring all other parameters. There exist a set of parameter hulls, one for each value of the parameter under study. Since overall convex hull always lies beneath all parameter hulls, the area between a parameter hull and the overall convex hull denotes the amount of lost efficiency due to setting the parameter to that fixed value. Therefore, if one had to set the parameter to one specific value, one should choose the value that corresponds to the parameter hull with the minimum area difference. A small minimum area suggests that there exists one best default value for the parameter under study. A large minimum area indicates it is important to re-adjust the parameter to optimize performance.

5.2 Effect of Parameters in Churn Intensive DT-Chord

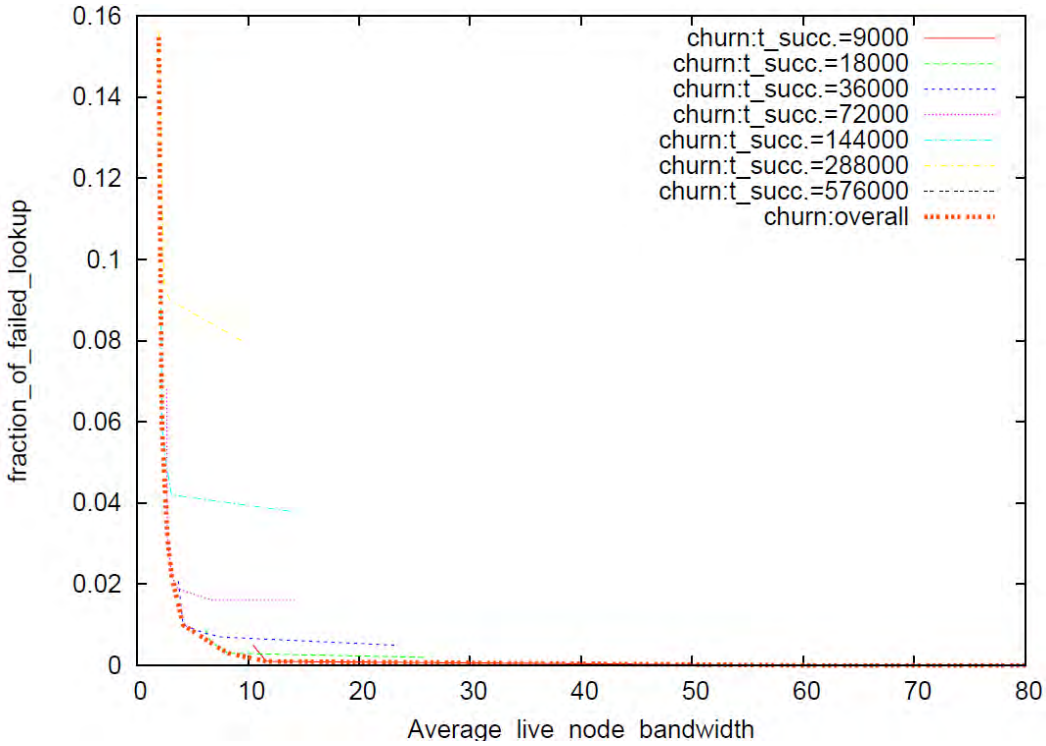
5.2.1 Effect of Successor Stabilization Interval

Base Chord and enhanced DT-Chord separately stabilizes finger and successors. The most important parameter, in terms of failure rates, is the successor stabilization interval (t_{succ}). This parameter governs how often a node checks that its successor is still alive, and thus the amount of time it takes a node to realize that its successor is dead and should be replaced with the next live node in ID space. The reason that t_{succ} has the largest effect on failure rate is that the correctness of the lookup protocol depends only on successor pointers, and not on the rest of the Chord routing table [14]. Thus it is enough to stabilize only the successors frequently if a low lookup failure rate is required. Faster successor stabilization interval result in wasted bandwidth while slower rates result in a greater

number of timeouts during lookups. In the Figure 5.1 and 5.2, overall convex hull and the parameter convex hulls for stabilization interval (t_{succ}) showing failure rate vs. bandwidth tradeoffs and successful lookup latency vs. bandwidth tradeoffs of all t_{succ}, under the churn intensive workload. A parameter convex hull that lies towards the bottom left of another hull indicates that its corresponding parameter can be tuned to have lower failure rate while consuming the same amount of bandwidth.

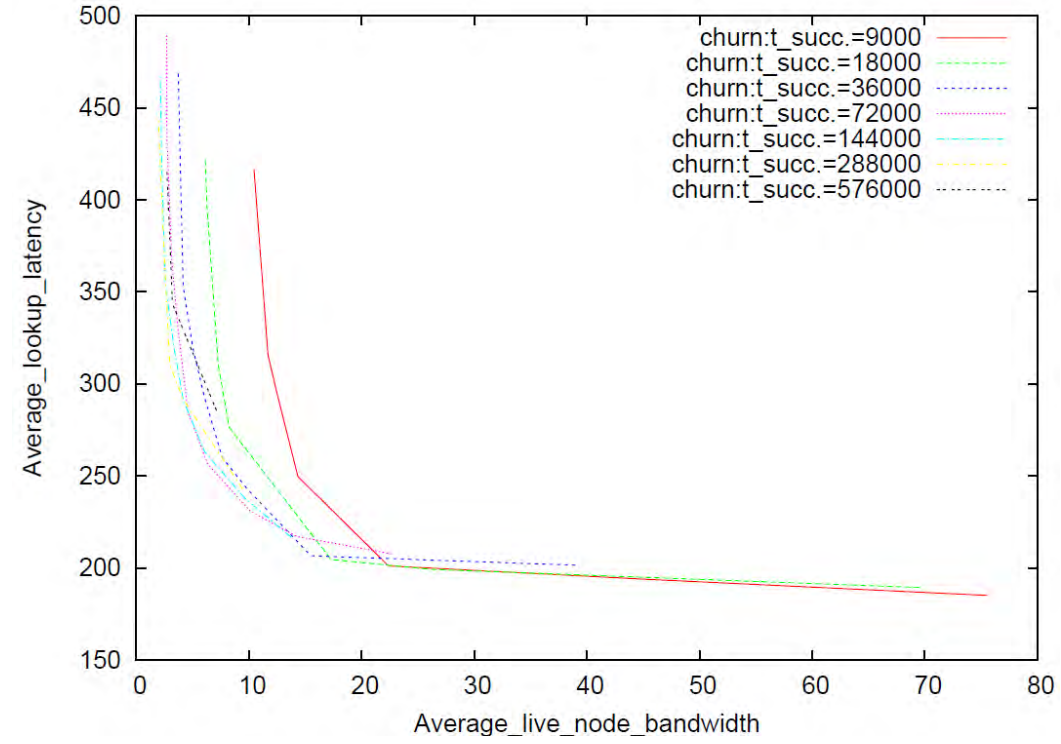


(a)

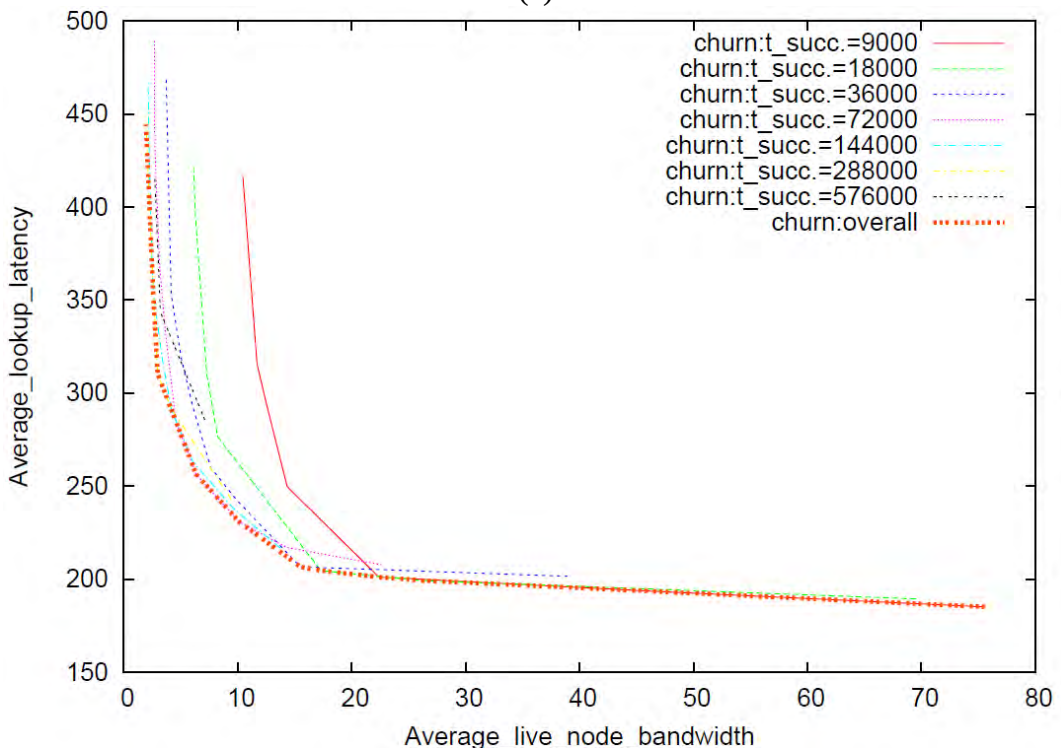


(b)

Figure 5.1: (a) Parameter convex hulls and (b) overall convex hull showing failed lookup vs. bandwidth tradeoffs of all t_{succ}, under the churn intensive workload.



(a)

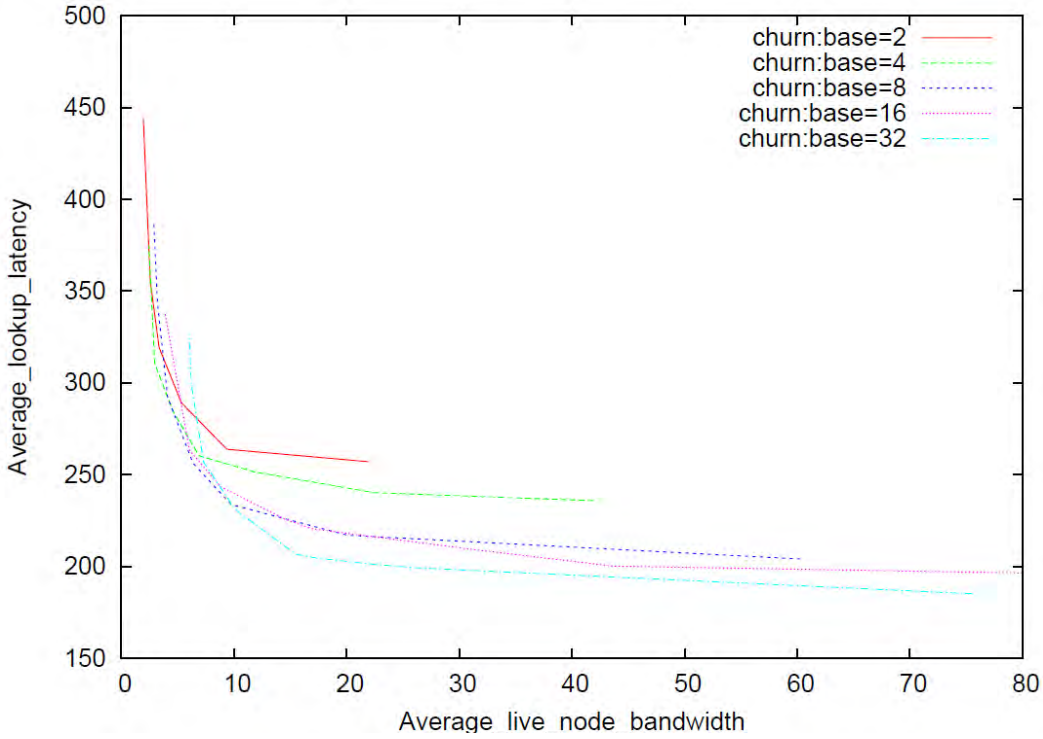


(b)

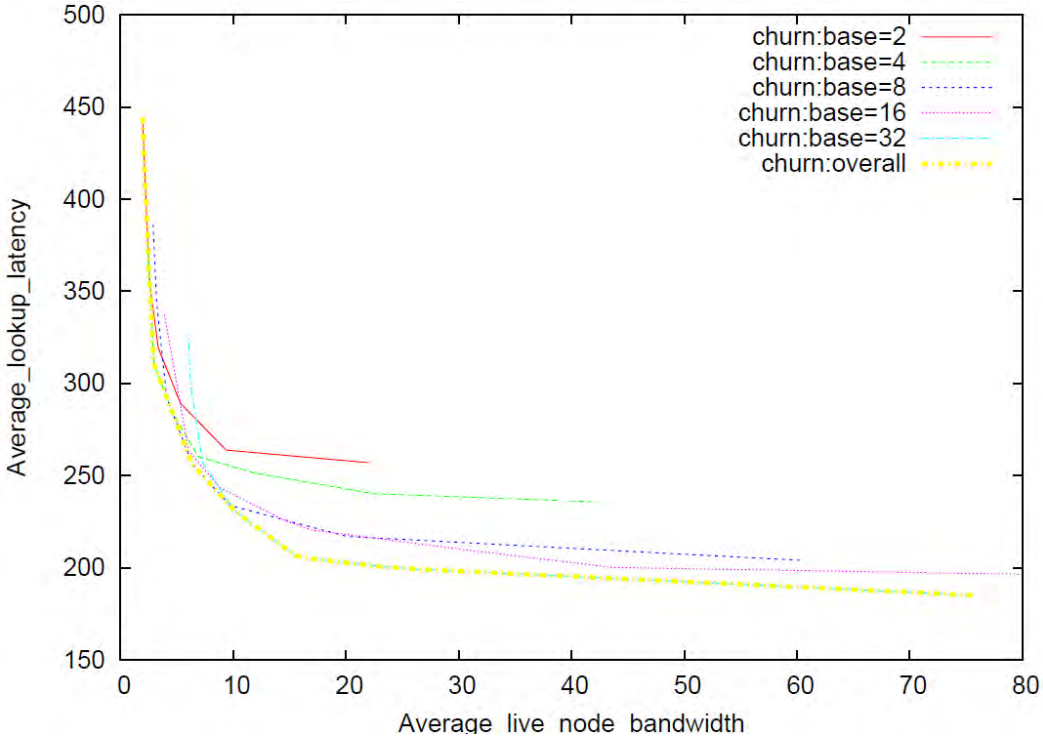
Figure 5.2: (a) Parameter convex hulls and (b) overall convex hull showing successful lookup latency vs. bandwidth tradeoffs of all t_{succ}, under the churn intensive workload. Each line traces the convex hull of all experiments with a fixed successortimer, t_{succ} value while varying all other parameters.

5.2.2 Effect of Routing Table Size

DT-Chord have base (*b*) as the parameter that is most in need of tuning for best lookup latency. Base controls the number of routing entries each node keeps and bigger bases lead to bigger routing tables with *(b-1) ($log_b n$)* entries. Figure 5.3 shows DT-Chord's overall convex hull as well as its parameter hulls for different base values. At the left side of the graph, where the bandwidth consumption is small, the parameter hull for $b = 2$ lies on the overall convex hull which means smaller bases should be used to reduce stabilization traffic at the expense of higher lookup latency. When more bandwidth can be consumed, larger bases lower the latency by decreasing the lookup hop-count. Expanding a node's routing table is more efficient than other alternatives at using additional bandwidth. For example, one competitive use of extra bandwidth is to check for the liveness of routing entries more frequently as doing so would decrease the likelihood of lookup timeouts. However, when routing entries are "fresh enough", spending extra bandwidth to further reduce the already very low lookup timeout probability has little impact on the overall latency. Instead, when the routing table is fairly fresh, a node should seek to reduce lookup latency by using additional bandwidth to expand its routing table for fewer lookup hops.

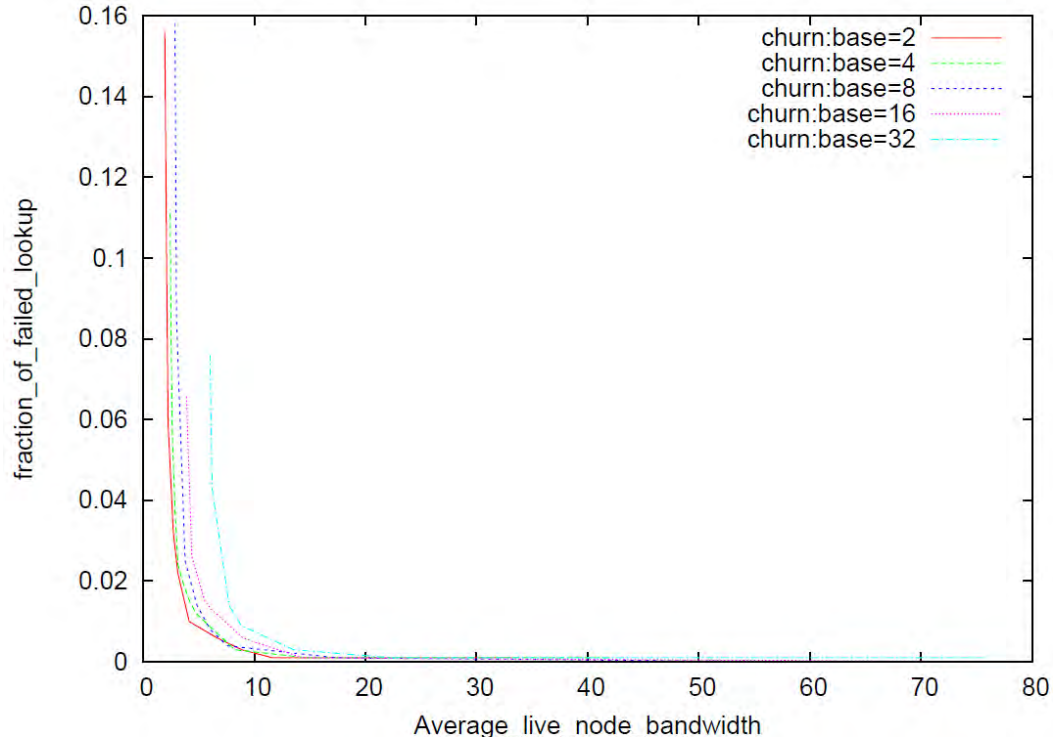


(a)

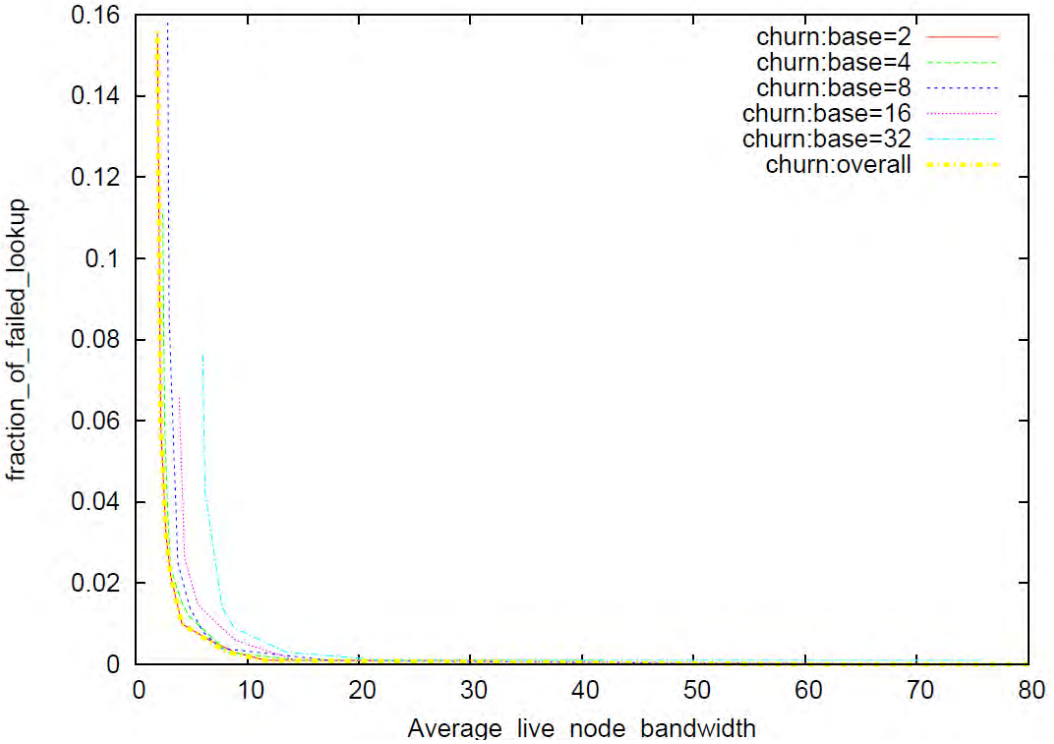


(b)

Figure 5.3: (a) Parameter convex hulls and (b) overall convex hull showing successful lookup latency vs. bandwidth tradeoffs of all base *b*, under the churn intensive workload. Each line traces the convex hull of all experiments with a fixed base *b* value while varying all other parameters.



(a)

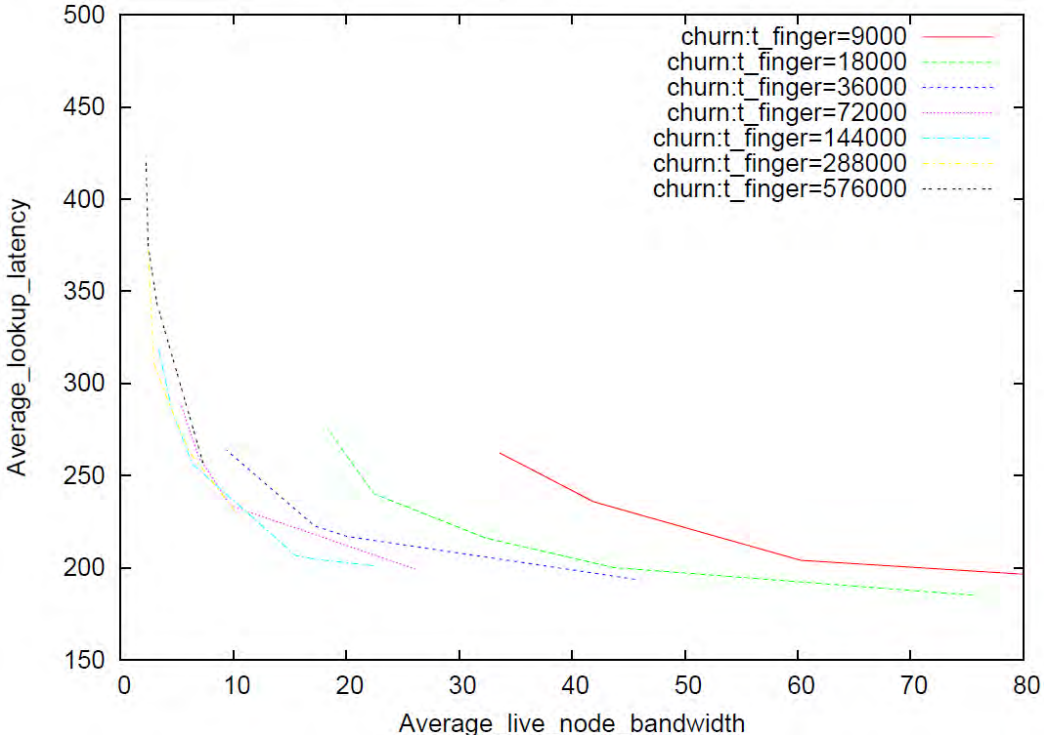


(b)

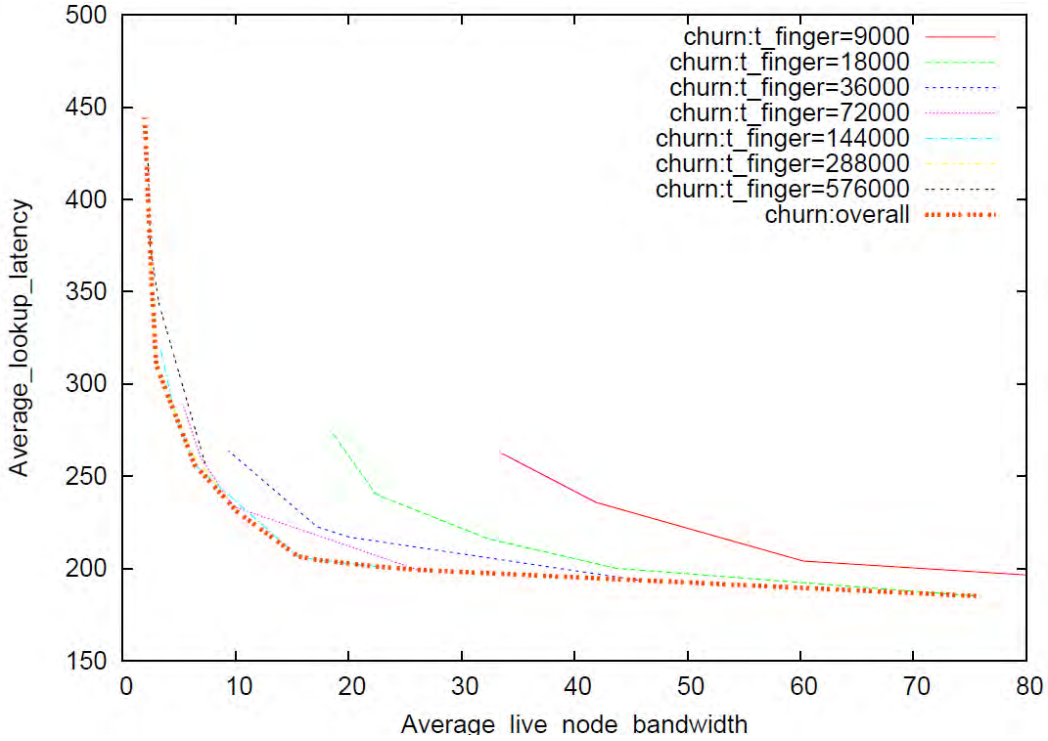
Figure 5.4: (a) Parameter *base* convex hull and (b) overall convex hull for lookup failure rate vs. bandwidth tradeoffs.

5.2.3 Effect of Routing Table Refresh Rate

The finger stabilization interval affects performance without affecting success rate, so its value must be varied to achieve the best tradeoff. Faster finger stabilization results in lower lookup latency due to fewer timeouts, but at a higher communication cost. A node should expand its routing table when the existing routing entries are already fresh. What is a good freshness threshold for routing tables? We know, routing entries' staleness is bounded by the finger stabilization interval. Figure 5.5 shows the parameter hulls for different stabilization values as well as overall convex hull. The best value for t_{finger} (144s) corresponds to a parameter hull that approximates the entire overall convex hull. Since parameter hulls are computed by exploring all other parameters including base (b), a less attractive parameter hull indicates the efficiency loss by setting t_{finger} to a wrong value. Making routing entries fresher than necessary (t_{finger} = 9 sec) results in a less efficient parameter hull as the extra bandwidth is wasted on checking the already sufficiently up-to-date routing entries as opposed to expanding a node's routing table. Allowing routing entries to become too stale (t_{finger} = 576s) also dramatically decreases the efficiency of the convex hull as stale routing entries lead to too many timeouts which can not be compensated by routing via fewer hops with a larger routing table.

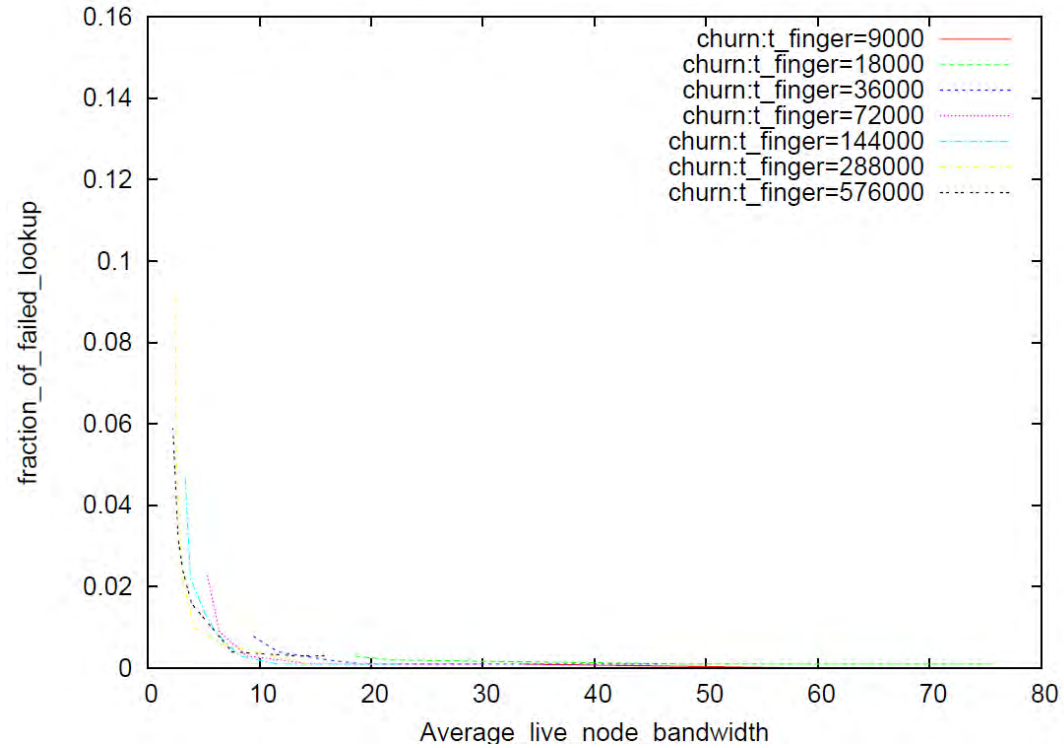


(a)

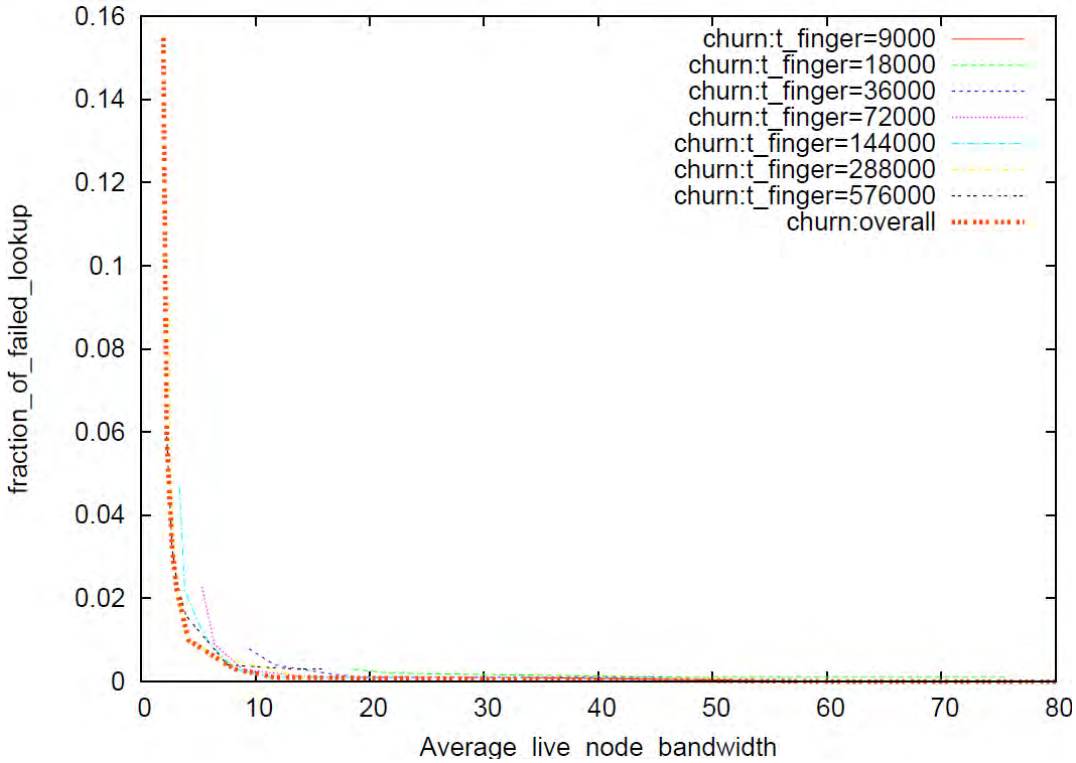


(b)

Figure 5.5: (a) Parameter convex hulls and (b) overall convex hull showing successful lookup latency vs. bandwidth tradeoffs of all *fingertimer*, t_{finger}, under the churn intensive workload. Each line traces the convex hull of all experiments with a fixed *fingertimer* value while varying all other parameters.



(a)



(b)

Figure 5.6: (a) Parameter convex hull and (b) overall convex hull showing failed lookup rate vs. bandwidth tradeoffs for all *fingertimer*, t_{finger}.

5.3 Effect of Parameters in Lookup Intensive DT-Chord

In the lookup intensive workload, each node issues and forwards much more lookup messages during its lifetime and hence the amount of churn is relatively low. Thus, it is more efficient to keep a larger routing table for fewer lookup hops when the amount of stabilization traffic is low compared to the amount of lookup traffic. Furthermore, fewer lookup hops translate into a large decrease in forwarded lookup traffic, given the large number of lookups. In Figure 5.7 Compared with churn intensive workload, convex hull for lookup intensive workload is relatively flat. Unless otherwise noted, from this section, all figures are for simulations done in the lookup intensive workload of DT-Chord consists of 1024 nodes.

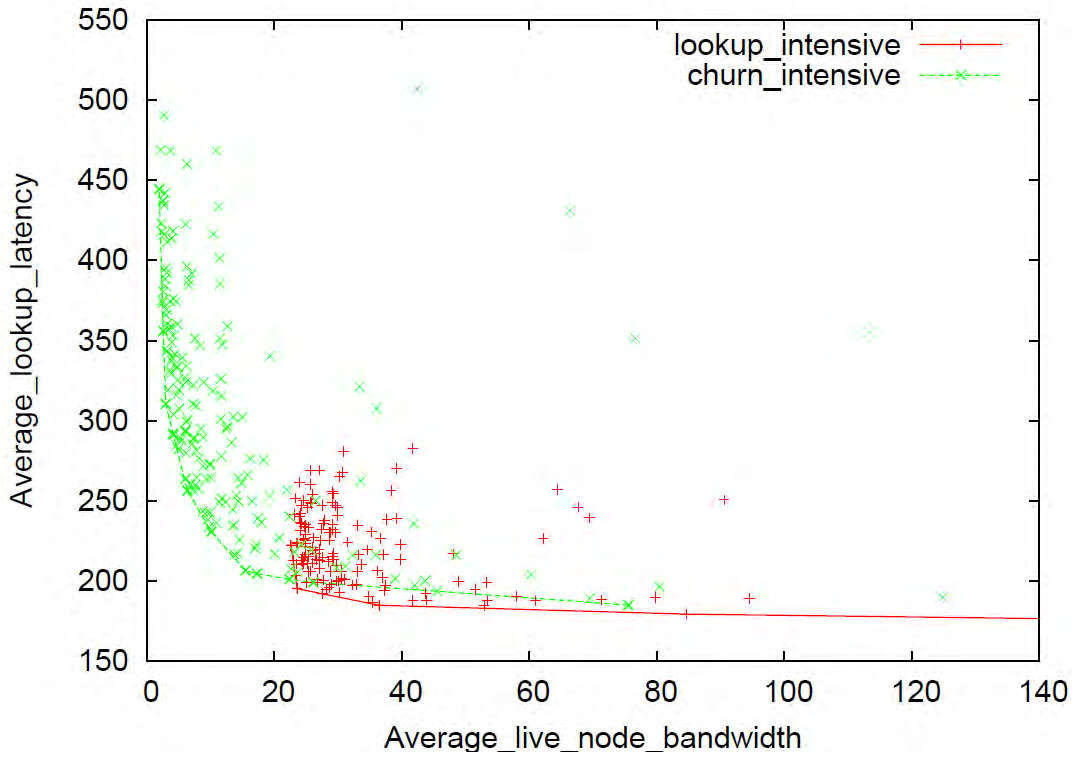


Figure 5.7: Overall convex hulls for successful lookup latency vs. bandwidth tradeoff of DTChord under the lookup intensive and churn intensive workload with network size 1024.

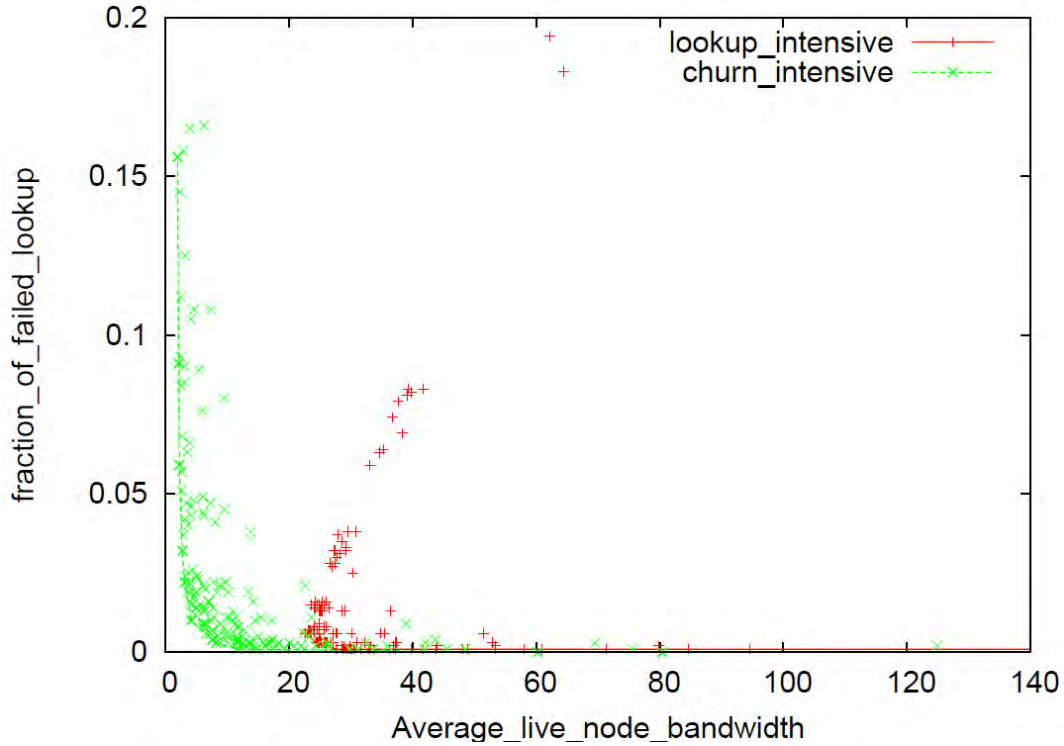
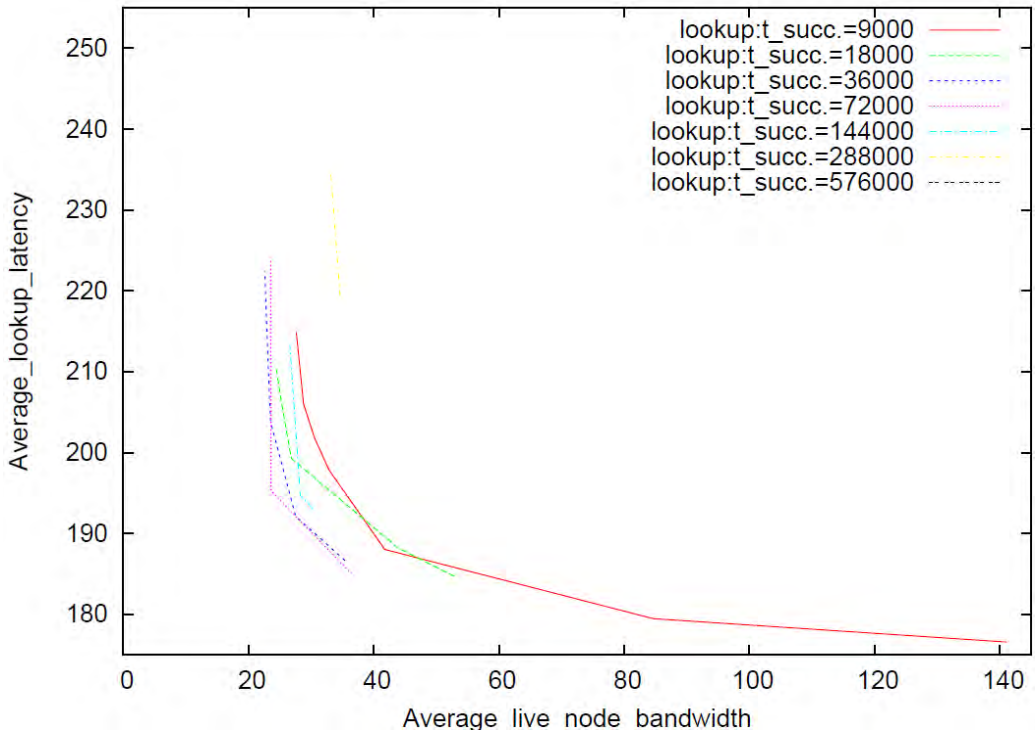
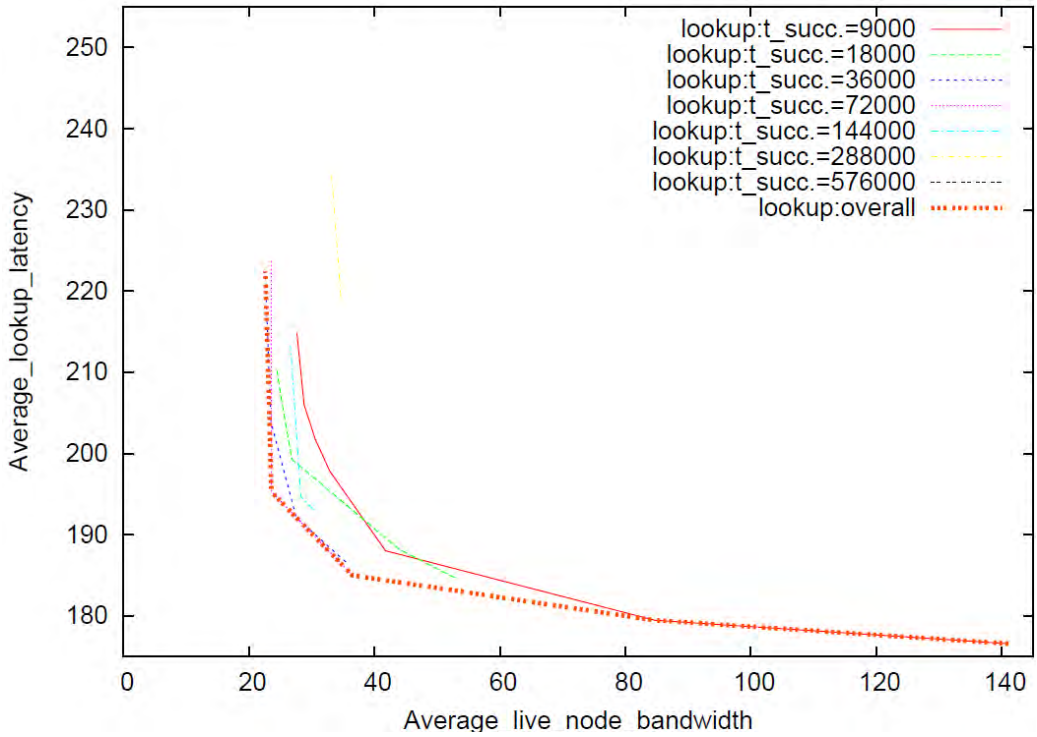


Figure 5.8: Overall convex hulls for lookup failure rate vs. bandwidth tradeoff of DTChord under the lookup intensive and churn intensive workload with network size 1024.

5.3.1 Effect of Successor Stabilization Interval in Lookup Intensive Workload

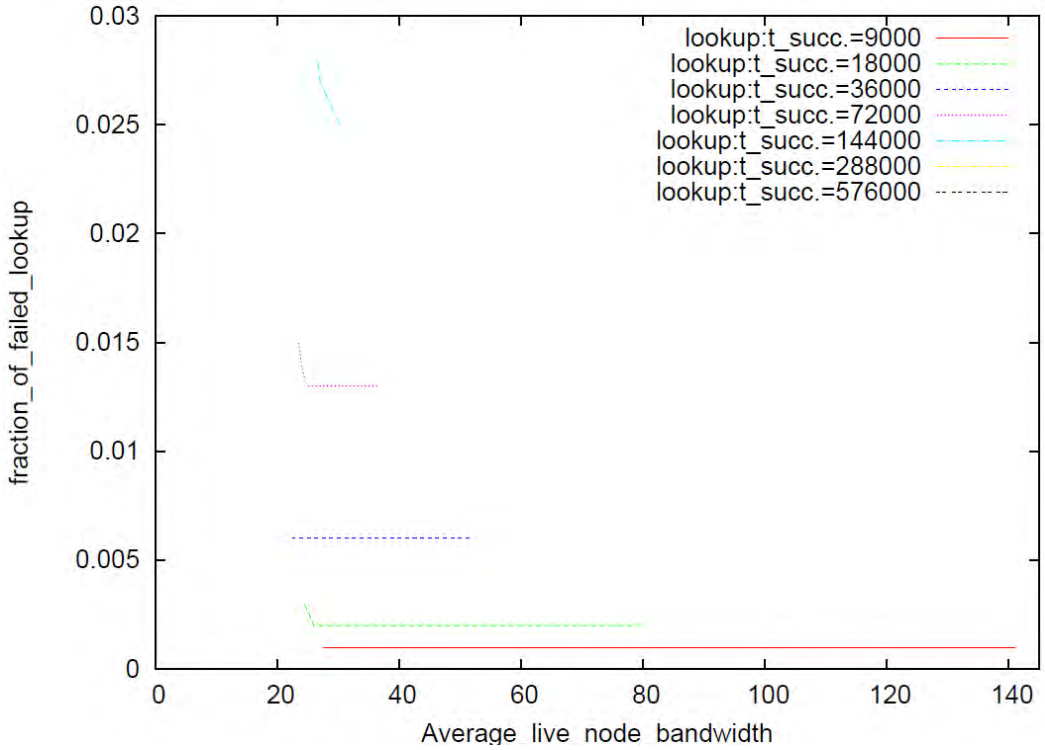


(a)

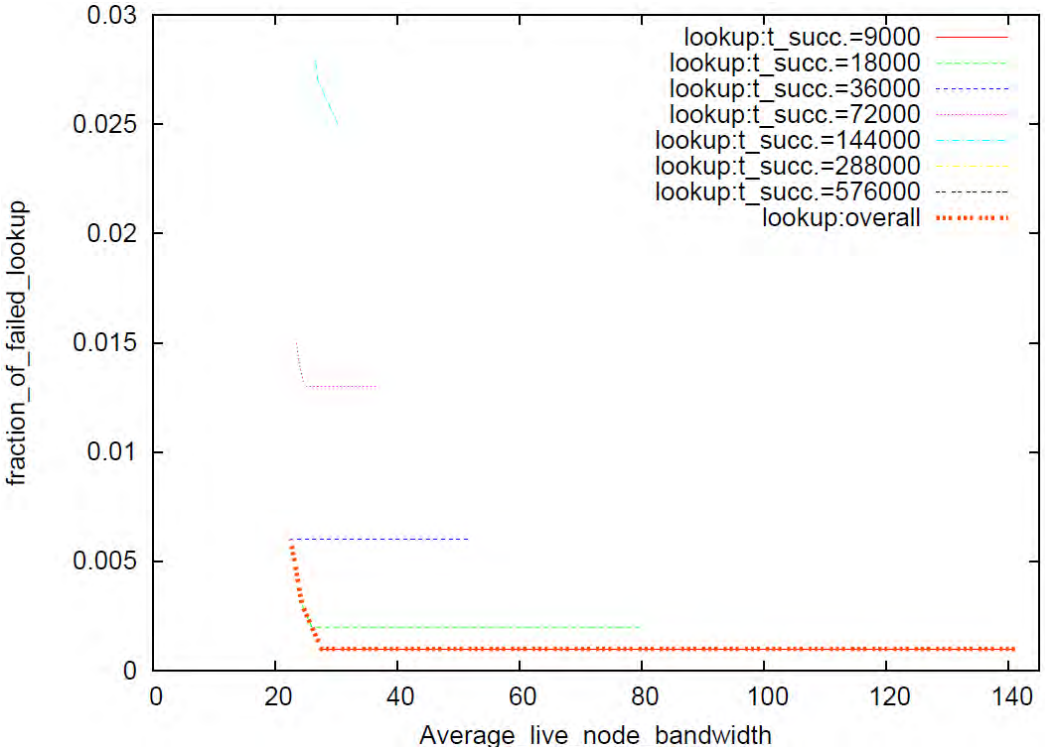


(b)

Figure 5.9: (a) Parameter convex hulls and (b) overall convex hull showing successful lookup latency vs. bandwidth tradeoffs of all t_{succ}, under the lookup intensive workload.



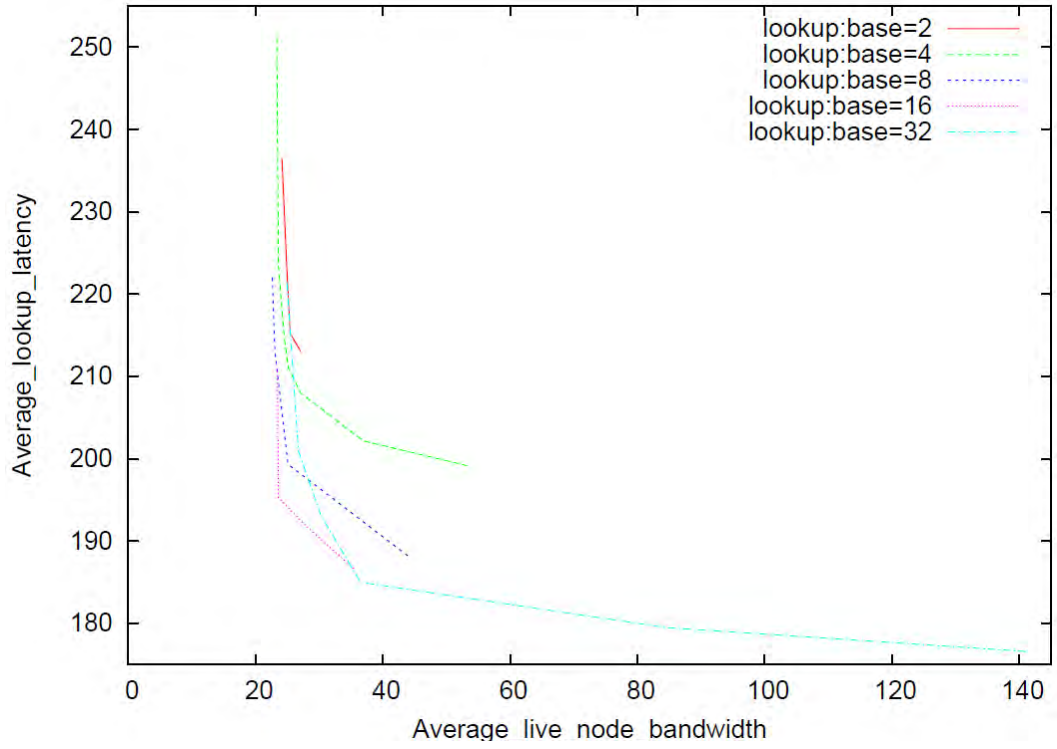
(a)



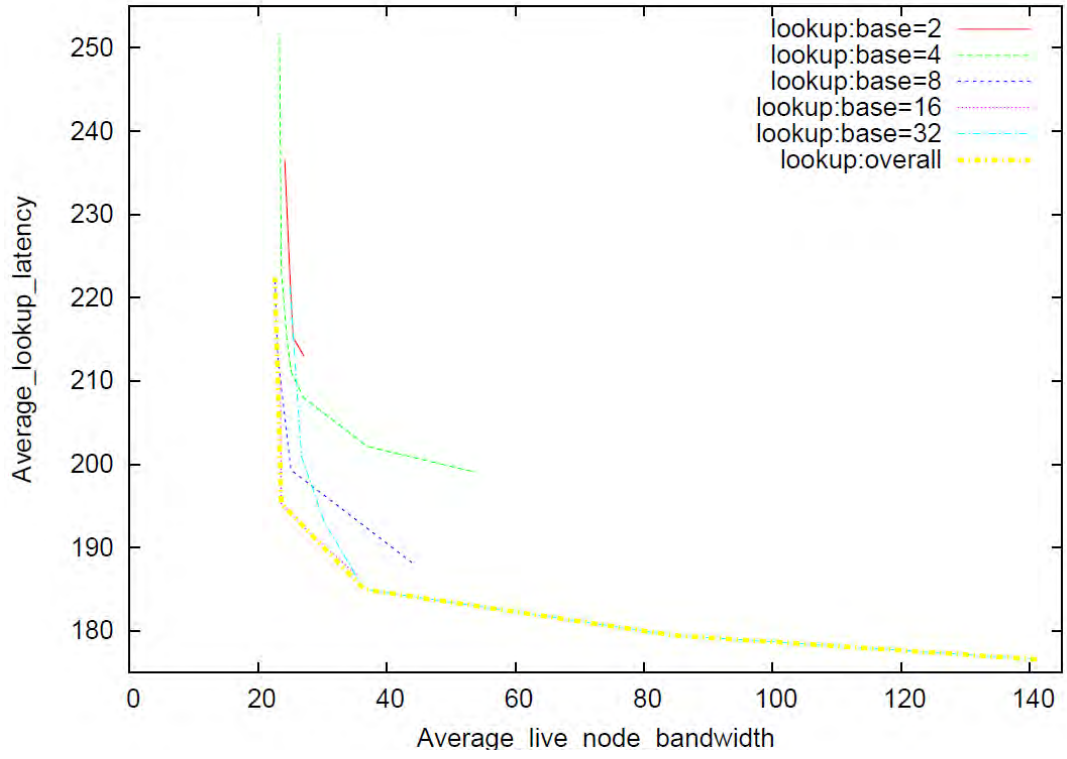
(b)

Figure 5.10: (a) Parameter convex hulls and (b) overall convex hull showing lookup failure rate vs. bandwidth tradeoffs of all successortimer $t_{successor}$, under the lookup intensive workload. Each line traces the convex hull of all experiments with a fixed base *b* value while varying all other parameters

5.3.2 Effect of Base in Lookup Intensive Workload

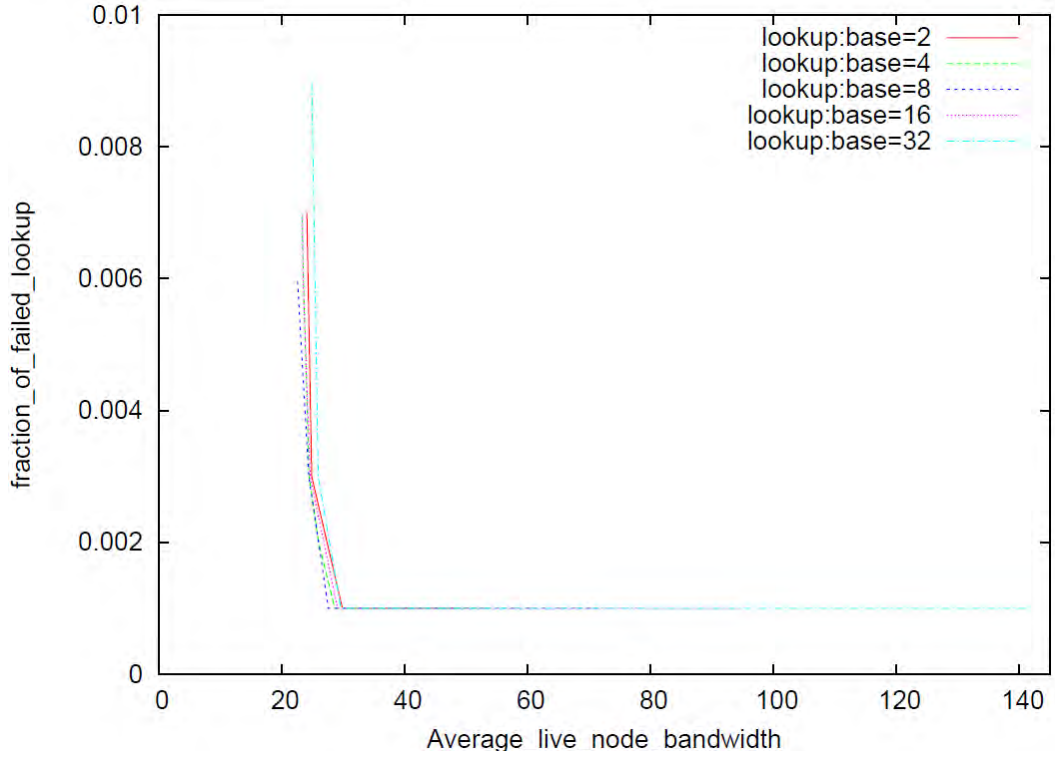


(a)

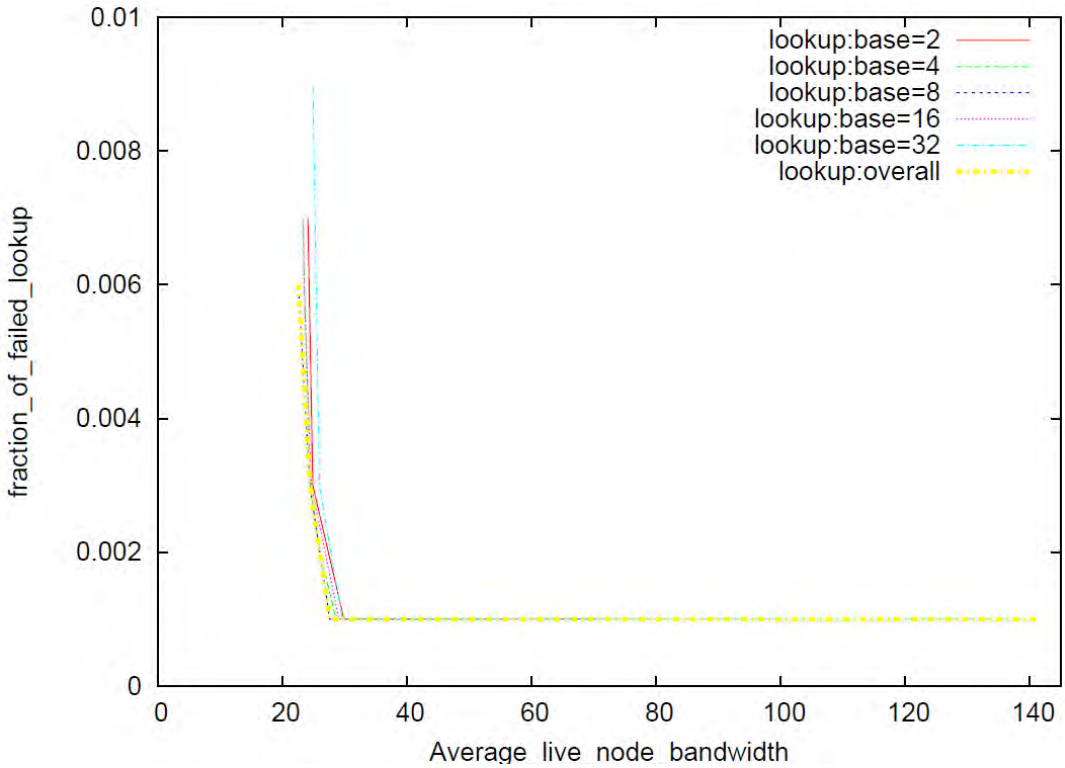


(b)

Figure 5.11: (a) Parameter convex hulls and (b) overall convex hull showing successful lookup latency vs. bandwidth tradeoffs of all base *b*, under the lookup intensive workload.



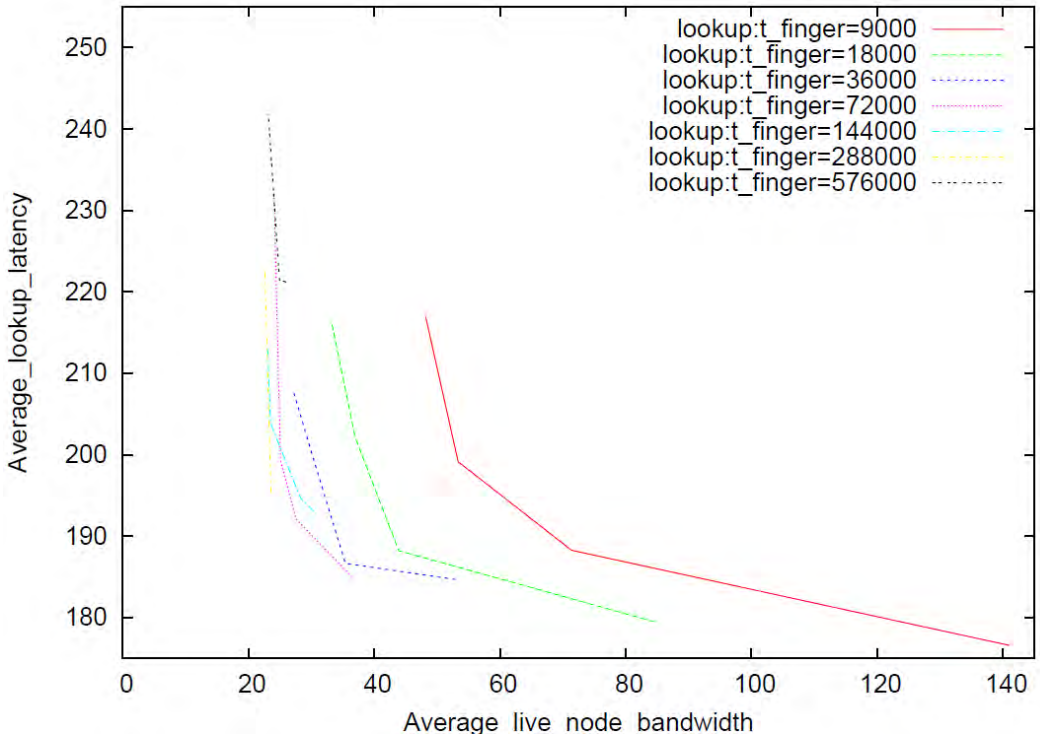
(a)



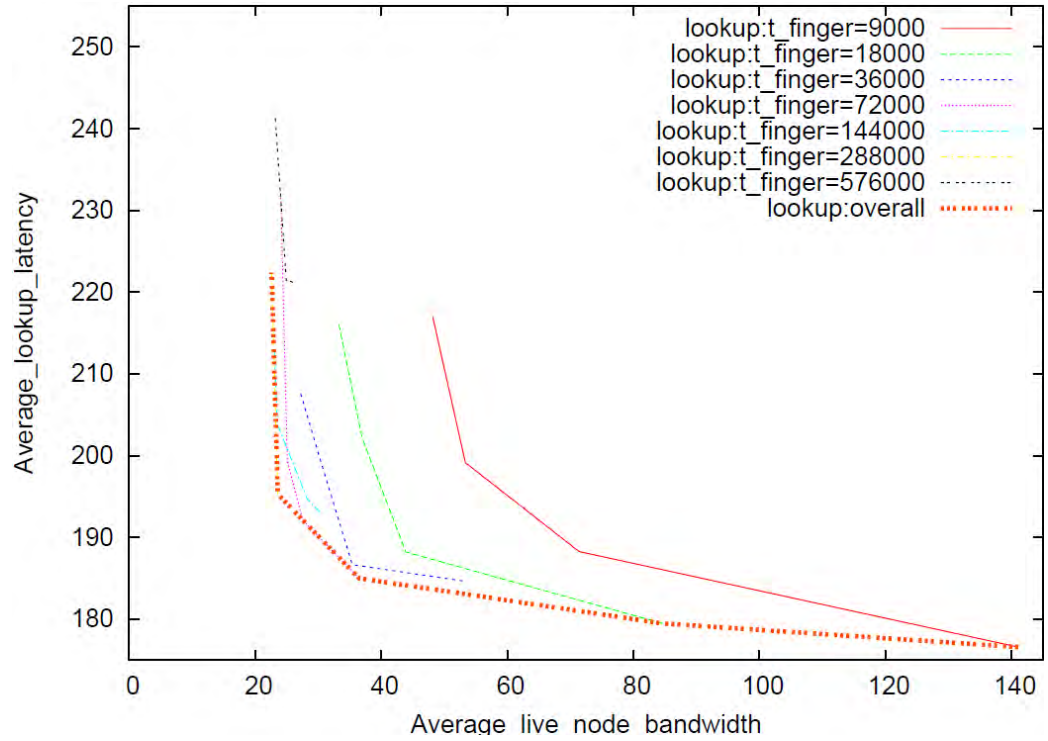
(b)

Figure 5.12: (a) Parameter *base* convex hull and (b) overall convex hull for lookup failure rate vs. bandwidth tradeoffs.

5.3.3 Effect of Fingertimer in Lookup Intensive Workload

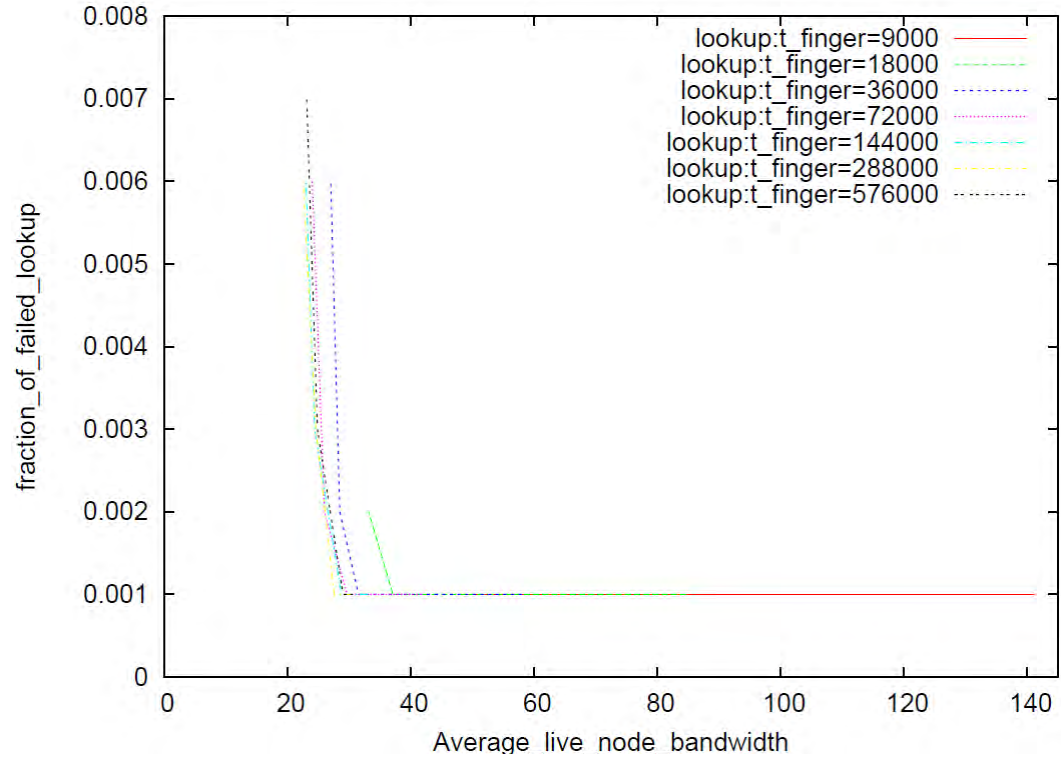


(a)

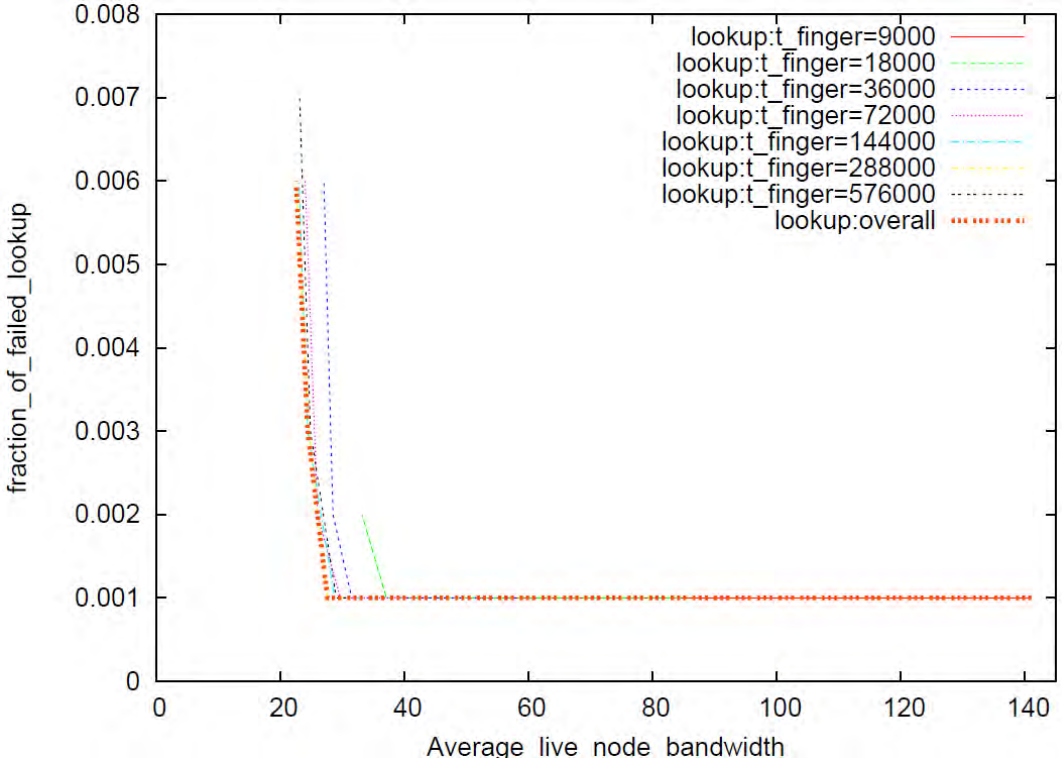


(b)

Figure 5.13: (a) Parameter convex hulls and (b) overall convex hull showing successful lookup latency vs. bandwidth tradeoffs of all *fingertimer*, t_{finger}, under the lookup intensive workload. Each line traces the convex hull of all experiments with a fixed *fingertimer* value while varying all other parameters.



(a)



(b)

Figure 5.14: (a) Parameter convex hull and (b) overall convex hull showing failed lookup rate vs. bandwidth tradeoffs for all *fingertimer*, t_{finger}.

5.4 Summary

We evaluate how efficiently different design choices use additional bandwidth for better lookup performance. In a real deployment, the protocol designer or deployer would have to tune the parameters manually to find the best values, or settle for default values. DT-Chord in particular can use its bandwidth quite efficiently and achieves low lookup latencies at little cost. This behavior appears to be due to its neighbor selection approach and giving priority to stabilizing successors over fingers when bandwidth is limited, since correct successors are all that is needed to ensure correct lookups. By focusing its limited stabilization traffic on this small, constant amount of state (as opposed to its full $O(\log n)$ state), DT-Chord is able to maintain correctness.

Table 5.1 summarizes the insights from the preceding sections. The best use for extra available bandwidth is for a node to expand its routing table. It is important to bound the staleness of routing entries and there seems to be a best freshness threshold under a given churn rate.

Table 5.1: The effect of successortimer, fingertimer, and base in DT-Chord.

	Average lookup latency	**Fraction of failed lookup**
successortimer	Faster successor stabilization interval results in wasted bandwidth and decreases average look up latency while slower rates result in a greater number of timeouts during lookups.	Largest effect on failure rate. The correctness of the lookup protocol depends only on successor pointers, and not on the rest of the Chord routing table.
base	Larger bases lower the latency by decreasing the lookup hop-count. Although this improvement comes at the cost of bandwidth.	Does not affect success rate.
fingertimer	Faster finger stabilization results in lower lookup latency due to fewer timeouts, but at a higher communication cost.	Does not affect success rate.

Chapter 6

Conclusion

We have presented DT-Chord, a DHT protocol for delay tolerant network, with a design that can be adjusted to reflect current operating environments and a user-specified bandwidth budget. By selecting lowest delay neighbor, DT-Chord achieves low lookup latency. The baseline Chord protocol performance and the enhanced DT-Chord performance are compared with extensive simulation. DT-Chord outperforms Chord in terms of bandwidth consumption, delay reducing, and number of successful lookup.

Evaluating DHT protocol over DTN in the presence of churn is a challenge. Methodologies developed for static networks can be misleading, since they don't account for the resources consumed to obtain low latency. We introduce a performance vs. cost analysis that explicitly accounts for the network bandwidth a DHT consumes to achieve better lookup performance. We incorporate features to improve lookup performance at extra communication cost in the face of churn in DTN. It is misleading to evaluate the performance benefits of an individual design choice alone because other competing choices can be more efficient at using bandwidth. We present protocol designers with a methodology to determine the relative importance of tuning different protocol parameters under different workloads (churn intensive or lookup intensive) and network conditions (*i.e.* delay, available bandwidth, network under churn). As parameters often control the extent to which a given protocol feature is enabled, our analysis allows designers to judge whether a protocol feature is more efficient at using additional bandwidth than others via the analysis of the corresponding protocol parameters. Furthermore, by remaining flexible in its choice of routing table size, DT-Chord can operate efficiently in a wide range of operating environments, making it suitable for use by developers who do not want to limit their applications to a particular network size, churn rate, or lookup workload.

6.1 Future Work

Our contributions improve the peer/resource lookup efficiency of Chord protocol over Delay Tolerant Scenarios. Lookup Routing issue in DTN is attracting more attention. DT-Chord can be applied to develop more realistic networks' application like mobility-assisted information diffusion and vehicle-based networks (VDTNs) application.

Bibliography

[1] Kevin Fall, “A Delay-Tolerant Network Architecture for Challenged Internets,” in *Proceedings of ACM SIGCOMM*, 2003, pp. 27-36.

[2] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K.Scott, and H. Weiss, “Delay-Tolerant Networking: An Approach to Interplanetary Internet,” in *IEEE Communications Magazine*, vol. 41, 2003, pp. 128-136.

[3] S. Jain, R. Shah, W. Brunette, G. Borriello, and S. Roy, “Exploiting Mobility for Energy Efficient Data Collection in Wireless Sensor Networks,” *ACM/Kluwer Mobile Networks and Applications (MONET)*, vol. 11, 2006, pp. 327-339.

[4] J. Partan, J. Kurose, and B. N. Levine, “A Survey of Practical Issues in Underwater Networks,” in *1st ACM International Workshop on Underwater Networks, in conjunction with ACM MobiCom*, Los Angeles, California, USA, 2006, pp. 17-24.

[5] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, “Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet,” *ACM SIGOPS Operating Systems Review*, vol. 36, 2002, pp. 96-107.

[6] W. Zhao, M. Ammar, and E. Zegura, “A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks,” in *The Fifth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2004)*, Roppongi Hills, Tokyo, Japan, 2004, pp. 187–198.

[7] A. A. Hasson, D. R. Fletcher, and D. A. Pentland, “A road to universal broadband connectivity,” in *Proceedings of Development by Design (dyd02)*, 2002.

[8] A. Rabagliati, “Wizzy digital courier–how it works,” April, 2004. http://www.wizzy.org.za/article/articleprint/19/1/2/.

[9] E. Brewer, M. Demmer, B. Du, M. Ho, M. Kam, S. Nedevschi, J. Pal, R. Patra, S. Surana, and K. Fall, “The case for technology in developing regions,” *IEEE Computer*, vol. 38, 2005, pp. 25–38.

[10] R. Y. Wang, S. Sobti, N. Garg, E. Ziskind, J. Lai, and A. Krishnamurthy, “Turning the postal system into a generic digital communication mechanism,” in *Proceedings of ACM SIGCOMM*, 2004, pp. 159–166.

[11] J. Ott and D. Kutscher, “Drive-thru internet: IEEE 802.11b for automobile users,” in *Proceedings of IEEE INFOCOM*, 2004, pp. 362–373.

[12] T. Small and Z. J. Haas, “The shared wireless infostation model: a new ad hoc networking paradigm (or where there is a whale, there is a way),” in *Proceedings of ACM MobiHoc*, 2003, pp. 233–244.

[13] L. B. Oliveira, I. G. Siqueira, D. F. Macedo, A. A. F. Loureiro, W. Hao Chi, and J. M. Nogueira, “Evaluation of peer-to-peer network content discovery techniques over mobile ad hoc networks,” in *Sixth IEEE International Symposium on World of Wireless Mobile and Multimedia Networks*, 2005, pp. 51–56.

[14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” in *SIGCOMM’01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, NY, USA: ACM, 2001, pp. 149–160.

[15] A. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems,” in *Middleware*, 2001, pp. 329–350.

[16] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, “Tapestry: A resilient global-scale overlay for service deployment,” *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, 2004, pp. 41–53.

[17] R. Sylvia, F. Paul, H. Mark, K. Richard, and S. Scott, “A scalable content-addressable network,” *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, 2001, pp. 161–172.

[18] S. Burresi, C. Canali, M. E. Renda, and P. Santi, “MeshChord: A location-aware, cross-layer specialization of chord for wireless mesh networks (concise contribution),” in *Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, 2008, pp. 206–212.

[19] C. Cramer and T. Fuhrmann, “Proximity neighbor selection for a DHT in wireless multi-hop networks,” in *Fifth IEEE International Conference on Peer-to-Peer Computing*, 2005, pp. 3–10.

[20] C. Cramer and T. Fuhrmann, “Bootstrapping chord in ad hoc networks not going anywhere for a while,” in *Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops*, 2006, pp. 168–172.

[21] C. Curt and F. Thomas, “Performance evaluation of Chord in mobile ad hoc networks,” in *MobiShare: Proceedings of the 1st ACM International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking*, 2006, pp. 48–53.

[22] A. Montresor, M. Jelasity, and O. Babaoglu, “Chord on demand,” in *Fifth IEEE International Conference on Peer-to-Peer Computing*, 2005, pp. 87–94

[23] T. M. Gil, F. Kaashoek, J. Li, R. Morris, and J. Stribling, P2Psim (MIT IRIS project). http://pdos.csail.mit.edu/p2psim/

[24] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai. “A survey of Peer-to-Peer network simulators,” in *Proc. of The Seventh Annual Postgraduate Symposium (PGNET)*, Liverpool, UK, 2006.

[25] P. Garcia, C. Pairot, R. Mondejar, J. Pujol, H. Tejedor, and R. Rallo. “PlanetSim: A new overlay network simulation framework,” *Lecture Notes in Computer Science (LNCS) Software Engineering and Middleware*, 2005.

[26] PeerSim: http://peersim.sourceforge.net/

[27] W. Yang and N. Abu-Ghazaleh. “GPS: a general Peer-to-Peer simulator and its use for modeling BitTorrent,” in *Proc. of International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2005, pp. 425-432.

[28] N. S. Ting and R. Deters. “3LS - a Peer-to-Peer network simulator,” in *Proc. of International Conference on Peer-to-Peer Computing (P2P)*, September 2003.

[29] M. T. Schlosser, T. E. Condie, and S. D. Kamvar, “Simulating a file-sharing P2P network,” in *Workshop on Semantics in P2P and Grid Computing*, 2002.

[30] K. P. Gummadi, S. Saroiu, S. D. Gribble, “King: Estimating latency between arbitrary Internet end hosts,” in *Proceedings of the 2002 SIGCOMM Internet Measurement Workshop*, 2002.

[31] P. K. Gummadi, S. Saroiu, and S. Gribble, “A measurement study of Napster and Gnutella as examples of peer-to-peer file sharing systems,” *Multimedia Systems Journal*, vol. 9, no. 2, Aug. 2003, pp. 170–184.

[32] K. P. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, I. Stoica, “The impact of DHT routing geometry on resilience and proximity,” in *Proc. of the ACM SIGCOMM*, 2003, pp. 381-394.

[33] Neurogrid: http://www.neurogrid.net/

[34] PlanetSim: http://planet.urv.es/planetsim/