NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

TAMBOUR: A SCALABLE AND ROBUST DHT PROTOCOL

A Thesis
Presented to
The Faculty of Graduate Studies
of
The University of Guelph

by
XIAO SHU

In partial fulfilment of requirements
for the degree of
Master of Science
April, 2009

© Xiao Shu, 2009

Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-56777-7
Our file Notre référence
ISBN: 978-0-494-56777-7

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

TAMBOUR: A SCALABLE AND ROBUST DHT PROTOCOL

Xiao Shu

University of Guelph, 2009

Advisor:

Professor Xining Li

Distributed Hash Table (DHT) is a new branch in the distributed system studies. Unlike majority of current file sharing peer-to-peer systems, DHT networks organize the peer-to-peer network in a structured manner and provide a simple lookup interface which is similar to a hash table.

In this paper, we present a new DHT protocol, Tambour, which bounds its communication overhead automatically and controls the lookup latency with high probability across a wide range of operating scenarios. Unlike many popular DHTs which employ a fixed routing table size, Tambour maintains a flexible routing table and tunes its size dynamically to improve lookup performance. It does not periodically probe its neighbours, but selects multiple nodes with high probability of availability and low latency to forward lookup messages to in a parallel manner.

In simulation evaluations, Tambour shows good lookup performance and robustness against unexpectedly high level of churn. We believe some techniques of Tambour can also be applied to many other DHTs to improve their performances.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Xining Li, for his kind assistance, wise advice and great patience. His wide knowledge in computer science and sharp vision of future trends provided a good basis for this thesis.

I am delighted to have Dr. Fangju Wang and Dr. Charlie Obimbo as my thesis committee members. I would like to thank them for their constructive comments, and for arranging time for the final oral examination. I would also like to thank Dr. Song Fei for chairing the examination.

My warm thanks are due to Jeremy Stribling and Jinyang Li who helped me build the simulation environment with *p2psim*. Their kind support and guidance have been of great value in this study.

I also wish to thank my friend and former roommate, Ryan McCullough, for proofreading my papers and helping me with my writing. I am really grateful for having such a wonderful friend.

Lastly, and most importantly, I wish to thank my parents, Changhong Shu and Husheng Wu, for their unconditional love and support. They started my interest in computer science when they enrolled me in programming classes at the age of eleven. To them I dedicate this thesis.

Contents

Acknowledgements	i
List of Tables	iv
List of Figures	v
1 Introduction	1
1.1 DHTs Overview	4
1.1.1 Structures	5
1.1.2 Advantages	8
1.1.3 Limitations	9
1.2 Research Motivation	10
1.3 Contribution	12
1.4 Thesis Organization	13
2 Related Work	14
2.1 DHT Geometries	14
2.1.1 $O(\log n)$ -hop Protocols	15
2.1.2 $O(1)$ -hop Protocols	17
2.1.3 Flexible Protocols	18
2.2 Optimization	19
2.2.1 Recursive vs. Iterative	19
2.2.2 Timeout Calculation	20
2.2.3 Neighbour Selection	21
2.3 Summary	21
3 Design Challenges	24
3.1 Routing Structure	26
3.2 Routing Update	30
3.3 Stability of Node	36

4	The Tambour Protocol	40
4.1	Routing Structure	41
4.2	Node State	42
4.2.1	Parallel Lookups	44
4.2.2	Routing Table Maintenance	46
5	Evaluation	48
5.1	Experimental Setup	48
5.2	Latency and Bandwidth	50
5.3	Lookup Workload	51
5.4	Churn Rate	53
6	Conclusion and Future Work	56
	Bibliography	59

List of Tables

2.1	A comparison of various DHT networks.	23
-----	---	----

List of Figures

1.1	DHT interface for structure P2P networks	3
1.2	A ring structure used in the primitive protocol.	6
3.1	Kleinberg's generalization of the Small World Phenomenon.	27
3.2	A $\frac{1}{x}$ distribution of neighbours in Tambour	28
3.3	The density of ID distances between a node and its neighbours after 30,000 lookups for random keys which matches a $1/x$ distribution well.	31
3.4	The cumulative distribution of node session duration in Gnutella net- work compared with an exponential distribution a Pareto distribution.	37
4.1	A host forwards multiple lookups to neighbours near the destination key	44
5.1	Tambour's latency vs. bandwidth trade off compared to Chord and OneHop, using a 3000-node network and a churn.	49
5.2	The average routing table size for Chord and Tambour as a function of the average, using a 3000-node network and a churn.	50
5.3	Tambour's lookup latency vs. bandwidth overhead tradeoff compared to OneHop and Chord.	52
5.4	The bandwidth overhead as a function of the lookup load for Chord, Accordion and Tambour.	53
5.5	The average lookup latency of Tambour, Chord and OneHop as a function of median node lifetime.	54
5.6	The average bandwidth consumption by Tambour, Chord and One- Hop as a function of median node lifetime.	55

Chapter 1

Introduction

It is always a challenge to design, implement, deploy and debug large-scale distributed systems as they operate in complex and changing environments. System designers' ultimate goal is to make a robust system that works correctly and efficiently across a wide range of operating conditions and is able to trade off certain performance merits to provide services continuously in unexpected circumstances without intervention.

Scalability is considered one of the most important characteristics of a distributed system because it is difficult to predict the ultimate size of the system in the design stage. Many popular systems just started out as experimental hobby projects and gain extremely popularity in short time. For example, the World Wide Web was fairly unknown by the public in early 1990s, however, with one decade nearly exponential growth, it is used on almost every computer connected to the Internet [WWW09]. A more recent example is the peer-to-peer technology based Internet telephony service Skype [Sky09b], which has just overthrown AT&T as the number one long distance call handler after 6 years since its introduction [Sky09a]. Such rapid expansion of system size comes with the growth of communication overhead, which could finally be in the way of letting the system growing even larger. System designers need to foresee this risk in the beginning.

Normally, the first step of systems design is to assume a certain size of the system and focus optimization for that scenario. As a result, some systems

produce best result with millions of nodes but sacrifice performance while the number of nodes in the system is not large enough, and others only work well for a smaller system but suffer from robustness and efficiency issues when the system grows unexpectedly large. The dilemma of choosing one design from the spectrum is what every system designer has to face – optimizing for either performance or scalability but not both.

With the unpredictable nature of system growth, an ideal system design should adjust itself automatically without forcing system builders to make too many assumptions about the deployment environment, especially the size of the system. The system would optimize itself to achieve better performance when it is small and maintain its robustness and efficiency as the network grows. The goal of this thesis is to study how to design and build a peer-to-peer system with the ability to adapt itself in various sizes of the system.

Peer-to-peer (P2P) networks are a distributed system design in which nodes self-organize into an overlay network and communicate with each other without specially designated servers. The exploitation of spare computing power and storage resources on end-user computers has long been a subject of interest and experimentation in distributed system study [SDRC82]. In recent years, the popularity of peer-to-peer systems does not only show the feasibility of building a large-scale distributed system with tens of thousands of heterogeneous hosts sharing resources on the Internet but also indicates the technical significance of P2P systems for today's computer and network environments.

The most interesting and unconventional feature of peer-to-peer networks are that, as more peer nodes join the system, they contribute more network and storage resources, increasing the aggregate system capacity. Thus, without investing in more powerful server infrastructure, peer-to-peer applications can potentially scale to many millions of nodes as the system grows. Popular commercial peer-to-peer applications include file sharing applications such as Gnutella [Gnu09], Kazaa [Kaz09], eMule [eMu09] and BitTorrent [Bit09], and Internet phone applications such as Skype [Sky09b] which shares similar technology with Kazaa. According to reports from Internet Service Providers, these P2P systems consume more than half of Internet traffic – sometimes even as

Figure 1.1: DHT interface for structure P2P networks

much as 75% [SW05]

How to locate data or resources among the numerous participating nodes is a major challenge and core problem in peer-to-peer system design. Unlike majority of current file sharing peer-to-peer systems, Distributed Hash Tables (DHTs) organize the peer-to-peer network in a structured manner and provide a simple lookup interface which similar to hash table [DZD⁺03]. As in Figure 1.1, Logically, each host in DHTs works like a bucket in classic hash tables, which stores and serves resources with certain keys. Each DHT provides a distributed lookup function as a replacement of the hash function in classic hash tables to locate the hosts, or say buckets, that serve certain key resources pairs. Once a host finds the host being responsible for certain keys by lookup function, loading or storing resources becomes a simple task between these two hosts. The elegant interface makes DHTs a potential universal building block for many distributed system applications.

There are various applications for DHTs, such as, distributed storage infrastructure with delivery guarantee, content distribution networks and web cache services. And DHTs are even capable of providing novel Internet architectures

such as overlay Internet routing [ZRM02]. Considering the wide range of deployment scenarios, it is crucial to create a robust system for DHT designers. In order to be scalable against rapid change of network size and unpredictable failures, DHTs must bound their communication overhead within the network capacity. This thesis provides an analysis of the design trade-offs DHTs and develops Tambour, a robust DHT that can adapt itself to offer reliable performance within a wide range of operating scenarios.

1.1 DHTs Overview

For DHT systems, the lookup function is the key to implementing a DHT. In fact, how to find the host that serves the given data item is the critical common problem in any P2P system.

Some traditional P2P systems come up with a central database, to map keys to hosts that store the corresponding values. For example, Napster [Nap09] maintains a list of song titles and hosts offering those songs at central Napster server, and BitTorrent [Bit09] and eDonkey2000 [eDo09] use such similar approach except that they allow each file or host to specify their own mapping servers. Centralized solutions are still popular among P2P file-sharing systems because of their simplicity, but they have poor scalability and suffers from single point of failure problem.

To achieve better fault-tolerance, some P2P systems avoid using central servers by moving the lookup process to decentralized and self-organized P2P hosts. Gnutella [Gnu09] employs a broadcast protocol which is continuously forwarding lookup requests to all known neighbours until the required resources are found. This flooding approach does not scale well either because of the unknown number of hosts that have to participate for each lookup request. FastTrack [Fas09] and its popular applications like Kazaa [Kaz09] handle such scaling problems by restricting flooding to volunteered “super-peers”, the hosts with high bandwidth and disk space. However, this scalability comes at the expense of resilience to failures of super-peers.

In Freenet [Fre09], the lookup query is forwarded from one host to another

based on an estimate of which host is most likely to store certain content. Although this unstructured search avoids blindly broadcasting requests, the query may not be guaranteed to reach the desired destination in a limited time. In addition, unpopular resources which are not well cached by other hosts could be hard to find or may simply disappear from the system.

1.1.1 Structures

DHT networks promote several features that distinguish them from traditional client-server services and previous P2P systems:

- *Decentralized*: Peers cooperate with each other to query, load and store resources in the systems without any centralized indexes.
- *Symmetric*: Every host takes exactly the same responsibility in providing lookup services. No host is supposed to be more important than others in order to improve the resilience of the whole system.
- *Structured*: The network topology is tightly coupled and resources are placed at precisely specified locations, so that queries can be efficiently routed to the hosts with desired contents.

The primitive protocol introduced with Chord [SMK⁺01] is a good example of how DHT networks implement above features. Here, each host is assigned a random m -bit identifier and a successor pointer recording the network address of the next host in the identifier space when it joins the system. If identifiers are represented as a circle of numbers from 0 to $2^m - 1$, then the successor host is the first peer clockwise from current host (See Figure 1.2). Each resource item also has an m -bit identifier: key k , which is generated from the resource by using secure hash function such as SHA-1. And the host whose identifier is equal to k or the first one clockwise from k will be responsible for storing and serving the resource with key k . For example, in Figure 1.2, host 56 maintains the data with key 54.

The lookup process is simple in this protocol. Each host forwards queries to its successor host until it reaches the host who knows where the key is. For

Figure 1.2: A ring structure used in the primitive protocol.

example, in Figure 1.2, host 8 will forward lookup to 14, 14 will forward to 21, and so forth. Eventually, the request arrives at host 51 who knows its successor 56 should have the key 54, then 51 will finalize the query by replying 8 with the address of 56. This process guarantees that the numeric difference between identifier of current host and the identifier of data will be shortened at least by one in each step, so lookup message will arrive to its desired destination for certain.

Although this protocol has the desired features: decentralized, symmetric and structured, it is not efficient enough for practical use. To design an effective DHT network protocol, there are two metrics that must be considered:

- *Hopcount*: Which is the number of network hops required for each lookup request. Usually, having smaller hopcount implies lower latency, faster lookup and better resilience since fewer hosts are involved in each request. The primitive protocol of Chord requires $O(n)$ hopcount for an n nodes network, which is too slow in real-life application.
- *Number of neighbours*: Having fewer neighbors means that less bandwidth

is consumed by maintaining the state of neighbours, but it also means poorer resilience under churn and larger hopcount [LSG⁺04]. In the case of the primitive protocol, each host maintains only one neighbour, namely, the “successor”. However, this protocol suffers from single-point-of-failure problem, and has high hopcount.

It is not possible to achieve both low hopcount and few neighbours at the same time [KK03], so P2P network protocols try to trade off between these two metrics to satisfy different situations. Many main stream protocols in DHTs research can balance between hopcount and number of neighbours to achieve better efficiency. Most of them can complete lookup process within $O(\log n)$ hops while maintaining the state of $O(\log n)$ neighbours, such as, Chord [SMK⁺01], Pastry [RD01], Tapestry [ZHS⁺04], Kademlia [MM02], and CAN [RFH⁺01]. Some protocols require only a constant number of neighbours, such as, Viceroy [MNR02] and Koorde [KK03].

These networks introduced above all employ a small number of neighbours for each host, which lead to multiple routing hops, typically $O(\log n)$. None of them try to maintain complete system membership information on every host, because their designers believe the bandwidth or memory requirement of such approach could be overwhelming for a large-scale membership. However, some research suggests that, in realistic deployment scenarios for P2P systems, maintaining complete membership information is possible and beneficial [RB04, BR03].

The reason is that, to provide guarantee of data retrieval, the DHT networks have to spend much bandwidth on replicating data which makes the membership maintenance cost minor. In other words, the feasibility of maintaining data requires membership to be stable enough that maintaining full membership is cheap. So even using a trivial protocol [RB04] maintaining full membership information on each host, the P2P systems still spends a small proportion of bandwidth on membership maintenance if membership is stable under the definition in [BSV03].

1.1.2 Advantages

With the brief introduction of the structures of DHTs, we now enumerate some of the advantages of DHTs.

- There is no centralized structure in DHTs. Every node is capable of performing lookup primitives, put, or get any keys, and the only requirement of joining the DHT for a new node is knowing one other existing node. Many real-life applications in which we want to deploy a DHT are completely decentralized by their nature. Allowing for no central point of organization in such environment is a great advantage.
- In most DHT designs, each node only needs to know $O(\log n)$ neighbours in the network. It is not necessary to maintain the status of the whole network. This feature of DHTs is important because a DHT node usually has to periodically probe its neighbours in order to detect their liveness; the bandwidth usage of the DHT thus scales linearly in the number of neighbours per node.
- For many DHTs, a node could choose other nodes as its neighbour from a large pool of candidates. Proximity neighbour selection (PNS) is the technique by which a node chooses within each group to minimize the network latency between it and its neighbours. Using PNS greatly improves the average latency of lookups which can be made constant in the size of the network. [DLS⁺04].
- DHT networks are robust and capable of survival under heavy churn. Even if each node in the network fails at probability $1/2$ at the same time, with high probability no disconnection will occur and the basic structure of the DHT will still hold [SMK⁺01].
- DHTs are almost completely self-organizing and self-maintaining. The only information a node requires to join a DHT is the address of another node in the network. This can be provided by user or from a public list of well-known nodes, and no other additional configuration information is necessary.

1.1.3 Limitations

DHT is an efficient and powerful distributed system scheme for many applications, such as index and cache services, but limited for data storage systems. As described by Blake and Rodrigues, there are three factors that limit the performance of a DHT: the bandwidth of each node, the total amount of data stored, and the membership turnover rate in the system [BR03]. Note that the partitioning of the key space in a DHT is entirely a function of the nodes that comprise the system at any given time. As such, whenever a new node joins the system or an existing node fails, the partitioning changes, and data must be moved so that get requests continue to be routed to the nodes storing the desired data.

DHTs keep multiple replicas of each data item for fault tolerance. When node leaves the network, it takes the data it has stored with it. To maintain the level of redundancy, the data has to be copied onto the remaining hosts. This problem is not unique to DHTs, but applies to any data storage system with replication mechanisms. Thus, any such system must either have a relatively stable membership, a large size of bandwidth, or store a small amount of data stored on.

On the other hand, the cost of nodes joining is unique to DHTs. Since DHTs map each data item to certain ID space by the distribution of node IDs, when a new node joins a system and changes the original ID space, the system has to move data to fit the new one. This cost cannot be avoided without changing the nature of DHTs, but since it is no larger than the cost of nodes leaving, it does not alter the range of applications and basic limitation of DHTs.

This analysis shows that to store a large amount of data in DHTs, nodes must be relatively stable and stay in the system for a long time, otherwise, it requires large bandwidth to maintain the accessibility of the data. Under some environment this is not a unreasonable requirement. For example, in the Coral caching system, each participating web server could presumably remain part of the system for months or longer [FFM04]. However, machines may crash or need to restart after updating, which means, each of these nodes might not remain continuously available for the entire time. This problem is that there is

no easy way to distinguish temporary failures of nodes from permanent ones, especially when such temporary failures could last for hours. If a temporary failure is detected and triggers recovery mechanisms, and soon the node goes back to normal, the bandwidth used during this unnecessary recovery is wasted.

In summary, there are two domains of applications where DHTs are suitable. First, a DHT is appropriate for providing a small amount of data across a large set of peers even if the membership of such system is not so stable. Examples of such applications include all those where a DHT is primarily used as an index, such the KAD file index network used by eMule [eMu09], and FeedTree, a collaborative RSS and Atom delivery system [SMPD05]. Second, a DHT can also be used to provide access to a large amount of data with delivery guarantee, but only if the nodes in such system stay for a long time and are relatively stable. OpenDHT [RGK⁺05], OverCite [SLC⁺06], and UsenetDHT [SMK08] are all good examples of this latter use of a DHT.

1.2 Research Motivation

One of the biggest challenge every peer-to-peer system has to cope with is churn: nodes continuously join and leave the system. Most popular peer-to-peer applications are built upon normal Internet hosts which only participate in the networks for short period of time. Studies of file sharing networks observe that the median time a node stays in the system ranges from tens of minutes to an hour depending on variant applications [GGG⁺03, BSV03, YTD06, ZDL07]. The major problem churn poses for DHT routing is that it causes routing tables to become out of date over time. When a node is dead or leaves the system, it creates an invalid routing entry for those nodes who know it as neighbour. It is not a good assumption that departing nodes will be able to notify their neighbours before leaving, and in many DHTs, nodes do not even know who have them as neighbours. Stale entries are not easy to detect, and they result in expensive lookup timeouts, since it takes multiple round-trip time for a node to determine that a lookup packet has been lost and re-route it through another neighbour. The overall end-to-end lookup latency is determined by

both hopcount and network delay at each hop.

There are various methods to reduce lookup latency and increase the accuracy of routing tables under churn. In general, these methods generate extra communication to get more information about the liveness of existing neighbours and new nodes in the network. However, bandwidth consumption could increase the size of a DHT's routing table which leads to higher cost of routing maintenance, and even congest the network. Certain mechanisms of failure detection and management could trigger a positive feedback of communication overhead and eventually overload the DHT network [RGRK04]. In order to be robust in scenarios when the network becomes unstable and churn rate increases rapidly, DHT networks should control their bandwidth use to avoid flooding the network with control messages.

Most DHTs proposed have a relatively fixed routing table size. They maintain routing table containing either $O(\log n)$ neighbours to achieve $(\log n)$ hops or $O(n^k)$ neighbours to achieve $(\log n)$ hops. As a result, these DHTs are optimized for a specific deployment scenario. For example, if the available bandwidth is abundant relative to the size and churn rate of the network, protocols like OneHop, which use large routing tables, could achieve the better lookup latency. However, when the system grows to millions of nodes or suffers from large degrees of churn, the communication overhead of OneHop protocol could overload the capacity of the network. In such scenario, protocols with logarithmic routing tables, such as Chord [SMK⁺01] or Pastry [RD01], fit better.

An good DHT protocol design, which provides scalable performance under various scenarios, should be capable of altering the size of its routing table dynamically with bounded communication overhead. Besides adaptive routing table, a DHT should have a bandwidth efficient way to detect neighbour failures and keep routing tables up to date. Most DHTs, for example, Chord and Pastry, rely on a periodic liveness checking mechanism by which a node pings each routing entry periodically to check its liveness. Once a neighbour node is found to be dead, the node starts searching for a replacement. Intuitively, the faster a node pings, the less likely it is that a lookup ends up to a dead node and causes timeout. However, periodic pinging generates overhead messages. The

more frequent a node pings, the more bandwidth it consumes.

Maintaining a relative fresh routing table by pinging entries is one of the methods to decrease the probability of lookup failures thereby reducing lookup latency. Another way is to issue parallel lookup messages to multiple next hop nodes so that, as long as at least one message reaches the next hop, the lookup process is a success. It is obvious that the more parallel lookups a node issues, the less likely it is that lookups will be affected by timeouts. Similar to the ping method, redundant parallel lookup messages generate communication overhead as well. In fact, no matter what kind of technique used in a DHT, it always require extra communication bandwidth to deal with churn. Therefore, the goal of a DHT could not simply be interpreted as to achieve lower lookup latency. We are interested in getting the best lookup latency efficiently with bounded bandwidth overhead. In other words, we want a DHT that could balance between latency and bandwidth automatically.

1.3 Contribution

In this thesis, we introduce a new DHT protocol, Tambour, which bounds its communication cost with a bandwidth budget and automatically control the probability of lookup failure in order to provide reasonable lookup latency across a wide range of operating scenarios. Because Tambour bounds the overhead of its communication while providing a predictable lookup latency, it is robust against rapid system size changes or unexpectedly high level of churn.

Many popular DHTs organize neighbours in structured manners with a fixed routing table size. Tambour uses a different approach, which dynamically tunes its table size to get the best lookup performance. It does not use periodical probe to test the availability for most of its neighbours, but maintains a large routing table and selects next hop node in an opportunist way. When the system is small and relatively stable, Tambour allows routing table to grow bigger; when the system suffers from high churn, Tambour controls the size of routing table and tries to achieve good lookup performance with high probability.

The problems that Tambour must solve in order to tune itself are how to

arrive at the best routing table size with the restriction of bandwidth budget and the stability of the node population, how to choose the most effective neighbours to place in the routing table, and how to divide the maintenance budget between acquiring new neighbours and checking the liveness of existing neighbours. Tambour solves these problems with a unique probability framework. Unlike many other protocols, it is flexible and not based on a rigid routing structure such as a de-Bruijn graph or hypercube that constrains the number and choice of neighbours. Instead, each node learns of new neighbours as a side-effect of ordinary lookups, but biases the learning nodes which are closer in ID space so that the density of its neighbours is inversely proportional to their distance in ID space from the node. This distribution allows Tambour to vary the table size while still providing the same worst-case scenario guarantees as traditional $O(\log n)$ -hop lookup protocols.

Most existing DHTs more or less rely on users to set up various parameters in order to tune the performance of the networks under different environments. Unfortunately, studies show that many of these parameters are in practice either left unspecified or deliberately misreported [SGG02]. Instead of relying on users, Tambour automatically provides the best lookup performance in a robust way and fits itself as the deployment environment changes.

1.4 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 summarizes related work. Chapter 3 describes the motives of the technical designs of Tambour and provides some theoretical analysis of the protocol. Chapter 4 introduces the implementation of Tambour. Chapter 5 evaluates Tambour and compares it to existing DHTs in simulation. Finally, Chapter 6 concludes and discusses future work.

Chapter 2

Related Work

There is a large amount of related work on designing robust distributed system. The design and applications of DHTs bear resemblance to early work in distributed data structure. One main difference of DHTs from these early work lies in its large scale and wide area deployment. Such a deployment scenario makes precise failure detections impractical. Furthermore, latency and bandwidth optimizations are crucial for good and robust performance over the wide area network. In this chapter, we review related work in DHT protocol designs and techniques for improving performance over the wide area network.

2.1 DHT Geometries

The routing pattern and structure of a DHT network is commonly called its *geometry* [GGG⁺03]. The term geometry in DHTs is only used to specify the graph formed by the relation of nodes, rather than the routing algorithm, which is the method of selecting next hop from neighbour list. Since DHTs share basically the same routing algorithm, geometry is a useful metric for distinguishing one DHT from another. In the following sections, we introduce the underlying geometries of some of the most popular DHT protocols.

2.1.1 $O(\log n)$ -hop Protocols

This category contains many main stream protocols in structured P2P network research because of their good balance between hopcount and number of neighbours. Most of them can complete lookup process within $O(\log n)$ hops while maintaining the state of $O(\log n)$ neighbours, such as, Chord [SMK⁺01], Pastry [RD01], Tapestry [ZHS⁺04], Kademlia [MM02], and CAN [RFH⁺01]. Some protocols require only a constant number of neighbours, such as, Viceroy [MNR02] and Koorde [KK03]. In the following, we briefly discuss these protocols with respect to their overlay network topologies.

Ring-like

Chord [SMK⁺01] is based on the primitive protocol introduced in Chapter 1. Topologically, they all map hosts on a ring by their identifiers. In addition to the successor pointer for each host, Chord maintains a list of neighbour pointers called the finger table to accelerate the lookup. The i th finger in the table of host a points the successive host to $a + 2^i$ on the identifier circle. The power-of-two structure of the finger table ensures that the node can always forward the query at least half of the remaining identifier space distance to the key. So $O(\log n)$ hops are enough for a lookup request.

To achieve robustness and correctness, Chord uses a successor list to keep track of the addresses of the next $\log n$ hosts immediately after current host in identifier space, and it repairs this list and finger table continuously using a stabilization algorithm. Further studies show Chord can keep its ideal state with an enhanced stabilization algorithm even when half of joined hosts depart the system simultaneously [LNBK02]. A more flexible finger selecting protocol was presented in [GGG⁺03] and [LSG⁺04], which makes Chord more efficient.

Tree-like

Tree-like protocols use the length of a common prefix as the distance of two identifiers rather than their numeric difference. Pastry [RD01] and Tapestry [ZHS⁺04] use similar routing protocols. In such system with m -bit identifier space, each host maintains a m/b -level routing table, where the i -th level hosts

share the same first $(i - 1) * b$ digit with the identifier of current host. Hosts always forward lookup messages to others which have longer common prefix with the key until no more digits can be resolved. Finally, when identifiers only differ in the last b digit, Pastry and Tapestry will switch to the primitive ring-like method to jump to the destination.

Kademlia [MM02] defines bitwise exclusive-or (XOR) of two identifiers as their topological distance. It works in the same way as the first step of Pastry and Tapestry in term of increasing the common prefix. However, when it comes to the last b digit, the XOR topology is still able to lead lookup message to its destination without additional structure. Because of its simplicity, several P2P file-sharing softwares implement Kademlia on the traditional P2P systems as their file indexing service, such as, eMule (based on eDonkey2000) [eMu09] and BitTorrent [Bit09].

Other

CAN [RFH⁺01] is a mesh of nodes in a virtual d -dimensional dynamically partitioned coordinate space. Each host keeps track of its d neighbours in each dimension. A key is mapped onto a point in the coordinate space, and is stored at the node whose zone contains the point's coordinates. The CAN algorithm guarantees deterministic discovery of an existing resource in $O(n^{1/d})$ steps. To achieve $O(\log n)$ -hop performance, CAN has to reset d to $\log n$ and re-map the d -dimensional space periodically.

Viceroy [MNR02] approximates a butterfly network, which is an $O(\log n)$ -hop protocol, but requires only constant expected number of neighbours. However, the limited degree may increase the risk of a network partition and reduce the fault-tolerance. Similar to the primitive protocol, Viceroy has a robustness issue. It explicitly assumes that events such as peers failing, joining and leaving the operation do not overlap.

Koorde [KK03] is a new DHT based on Chord and the de-Bruijn graphs. While inheriting the simplicity of Chord, Koorde meets various lower bounds, such as $O(\log n)$ hops per lookup request with only 2 neighbours per host, and $O(\log n / \log \log n)$ hops per lookup request with $O(\log n)$ neighbours per node.

2.1.2 $O(1)$ -hop Protocols

Structured P2P networks introduced in previous section employ a small number of neighbours for each host, which lead to multiple routing hops, typically $O(\log n)$. None of them try to maintain complete system membership information on every host, because their designers believe the bandwidth or memory requirement of such approach could be overwhelming for a large-scale membership. However, some research suggests that, in realistic deployment scenarios for P2P systems, maintaining complete membership information is possible and beneficial [RB04, BR03].

The reason behind is that, to provide guarantee of data retrieval, the structured P2P networks have to spend much bandwidth on replicating data which makes the membership maintenance cost minor. In other words, the feasibility of maintaining data requires membership to be stable enough that maintaining full membership is cheap. So even using a trivial protocol [RB04] maintaining full membership information on each host, the P2P systems still spends a small proportion of bandwidth on membership maintenance if membership is stable under the definition in [BSV03].

Gupta *et al.* proposed a one-hop routing scheme based on cluster to reduce background keep-alive message broadcast [GLR04]. They also presented a similar two-hop protocol for large scale systems of more than a few million nodes. Simulation shows that these algorithms can maintain routing information sufficiently up-to-date such that a large fraction of the queries will succeed without being re-routed.

Kelips [GBL⁺03] divides hosts into $\sqrt{n}$ groups. Each host monitors all other hosts in its group and 2 nodes from each foreign group. So if requested host is in the same group, only one hop is needed; if requested host is in foreign group, current host will forward lookup request either of the two known hosts in that group, and lookup will terminate in two hops.

Leong *et al.* proposed a $O(1)$ -hop protocol based on the primitive ring-like protocol as introduced in Chapter 1 [LL04]. In this protocol, hosts pass tokens along the ring and propagate join/departure events by piggybacking them on tokens. To achieve better efficiency, several tokens are passed on the ring in

parallel. Simulation shows that this method has better performance than Gupta *et al.*'s OneHop algorithm.

2.1.3 Flexible Protocols

In general, $O(\log n)$ -hop DHT protocols require less bandwidth to operate and scale better when the system is large or unstable. On the other hand, although $O(1)$ -hop DHTs consume more bandwidth, they are simpler and provide better lookup latency that $O(\log n)$ -hop DHTs cannot match. The DHT geometries above present a tradeoff between the bandwidth used in maintaining neighbor links versus the number of hops required to perform a lookup. Given the bandwidth constrain, an ideal DHT should maintain a large routing table like $O(1)$ -hop DHTs and fallback to smaller routing table as the system grows larger or becomes unstable. A Natural question thus arises as to whether it is possible to design a DHT geometry that adapts to the bandwidth available to deliver the fewest hops possible in any situation. Tambour is one of the DHTs that have such kind of ability. It could cope with different system size and churn level automatically by using a more flexible routing table. Besides Tambour, EpiChord [LLD04] and Accordion [LSMK05] are flexible in term of choosing the size of routing table as well.

EpiChord [LLD04] is a Chord-based lookup algorithm which removes the $O(\log n)$ -neighbour-per-node restriction to achieve significantly better lookup performance and resilience. Meanwhile, it uses a novel reactive routing state maintenance strategy that amortizes network maintenance costs into existing lookups by issuing parallel queries. EpiChord is able to achieve $O(1)$ -hop lookup performance under lookup-intensive workloads, and at least $O(\log n)$ -hop lookup performance under churn-intensive workloads even in the worst case. This technique can be used to improve the efficiency of most $O(\log n)$ -hop protocols.

Symphony [MBRI03] is the first DHT protocol based on small-world phenomenon [Kle00]. Though it does not have a dynamic routing table that changes size automatically, its underlying structure is flexible. Every node in a Symphony network can be a potential neighbour for every other node, as long as the routing entries in its routing table follow the small-world distribution. Tambour

uses this small-world distribution to populate its routing table as well. However, while Tambour can adjust its table size according to bandwidth budget and churn, the size of Symphonys routing table is a fixed protocol parameter given by user. Symphony acquires the desired neighbor entries by explicitly looking up identifiers according to a small-world distribution. Tambour, on the other hand, acquires new entries by both learning from existing neighbors during ordinary lookups and active exploration. Mercury [BAS04] is another DHT that also employs a small-world distribution. Mercury optimizes its tables to handle scalable range queries rather than single key lookups.

Accordion shares many similarities with Tambour. It uses small-world distribution for choosing neighbours and has a bandwidth budget given by user to bound the communication cost. Considering that most peer-to-peer protocol parameters are in practice either left unspecified or deliberately misreported, Tambour does require user to specify the bandwidth budget, but selects budget automatically based on network activity and churn rate. And in Tambour, this budget only applies to active exploration, ordinary lookups are always welcomed unless churn rate increases in large scale. Another difference is that Accordion uses budget to decide the best node for next hop while Tambour uses a probability framework based on both liveness and locality to select several neighbours for parallel lookup.

2.2 Optimization

The geometry of a DHT is one important component of its performance in practice. The routing algorithm used are often just as important. There are several practical optimization methods used across different DHTs.

2.2.1 Recursive vs. Iterative

For any of these $O(\log n)$ -hop protocols, there are two methods can be used to send lookup message:

- *Recursive*: The originator sends a lookup request to each successive node in the lookup path, and waits for the response before proceeding.

- *Iterative*: Each node in the lookup path directly forwards the query to the next node, and when query reaches the key's predecessor, the predecessor sends its successor list directly back to the originator.

Recursive lookup, used by many protocols, has lower lookup latency than iterative since each intermediate host can immediately forward the lookup before acknowledging the previous hop [SMK⁺01]. Iterative lookup is slower comparatively, however, it is easier to recover from host failures than recursive. To take the advantages of both lookup methods, some implementations including Tambour use recursive lookup by default since they are faster, but fall back on iterative lookup after failures [DLS⁺04].

2.2.2 Timeout Calculation

In any large distributed system, there are always some nodes that are slow (due to temporary load, faulty components, etc.) or crash down without warning. Furthermore, studies of existing peer-to-peer systems deployed on the Internet show even higher rates of failure than other similarly sized distributed systems [SGG03, MN05, YTD06]. If failures are common in the system, and detecting and routing around a failed node takes several seconds, the cost of failures will add up to the cost of a lookup and easily dominate the latency.

With continuous active probing of its neighbors in the graph, a DHT node can have a good estimation for the time it expects each neighbor to take to process a lookup message [RGK⁺05]. Messages that are not acknowledged within this period can quickly be resent along an alternate path.

If the timeout value is too small, it could cause false positives and waste bandwidth on retransmitting. Thus, message timeouts generally need to be some small multiple of average round-trip time, even so, however, a single timeout can add significantly to lookup latency. One approach to limiting the effect of timeouts on end-to-end lookup latency is to parallelize the lookup; by routing along several paths simultaneously, a DHT can use extra resources to increase the number of timeouts that must occur in order to stall a lookup [LLD04].

2.2.3 Neighbour Selection

Another important optimization technique is called proximity neighbour selection (PNS) which decreases lookup latency by choosing nearby hosts as routing table entries [DLS⁺04, GGG⁺03]. Some of the protocols, such as, Pastry, Tapestry and Kademlia, can use PNS because there are few constraints in the choice of routing entries: any host in the relevant portion of the identifier space is eligible. These protocols can learn nearby hosts by caching passing messages or by searching algorithm. An exhaustive search may improve lookup latency, but also consume network resources.

Both experiment [GGG⁺03] and theoretical analysis [DLS⁺04] show that PNS approximates 1.5 times the average round trip time in the underlying network and this result holds regardless of the number of hosts (and thus regardless of the number of hops). Actually, the last few hops of a lookup dominate the total latency. As a lookup gets close to the target key in identifier space, the number of remaining nodes that are closer to the key decreases, and thus, the latency to the nearest one increases on average.

There is another technique called proximity route selection (PRS) [GGG⁺03], where a lookup query was routed at each hop to the neighbor closest in network latency that made progress in the identifier space. It shows that PRS outperforms greedy routing in a simulation of Chord using a realistic network latency distribution, but they did not account for the additional cost of processing at each node. As the PRS route is likely to involve more hops than the greedy one, this per-hop processing cost can be significant in real deployments [RGK⁺05].

2.3 Summary

DHT networks represent a new generation of P2P systems. They offer a number of advantages over conventional P2P systems, such as, pure decentralized, guarantee for resources delivery, and resilience under churn. In this chapter, we discussed several current structured P2P network protocols: $O(\log n)$ -hop protocols which maintain fewer number of neighbours, $O(1)$ -hop protocols which use less hops for each query and flexible protocols that tune themselves to fit

operating environments. Table 2.1 shows the summary of these protocols. With the evolution of P2P systems, both types of protocols start to import features that were only on the other type of protocol before. Some $O(\log n)$ -hop protocols try to learn as many hosts as possible so that they can decrease lookup latency by PNS; some $O(1)$ -hop protocols implement $O(\log n)$ -hop protocols as their base structure to reduce maintenance overhead. Research shows that both types of protocols can produce similar performance results under certain parameters.

Protocols	Geometry	Hopcount	Neighbours	PNS	Notes
primitive ring (Chap. 1)	ring	$O(n)$	2	no	
Chord [SMK ⁺ 01]	ring (finger)	$O(\log n)$	$O(\log n)$	yes	Node uses finger table (shortcuts) to reduce hopcount.
Pastry [RD01]	tree	$O(\log n)$	$O(\log n)$	yes	Tree structure for accelerating and ring for correctness.
Tapestry [ZHS ⁺ 04]	tree	$O(\log n)$	$O(\log n)$	yes	Similar to Pastry.
Kademlia [MM02]	tree (XOR)	$O(\log n)$	$O(\log n)$	yes	Pure tree structure, no ring needed.
CAN [RFH ⁺ 01]	d -dimension	$O(dn^{1/d})$	$O(d)$	no	
Viceroy [MNR02]	butterfly	$O(\log n)$	$O(1)$	no	
Koorde [KK03]	de-Bruijn	$O(\log n)$	2	no	It uses the fewest neighbours, but it is rigid and does not support PNS.
OneHop [GLR04]	2-level hierarchy	1	$O(\frac{\log n}{\log \log n})$		It has a hierarchy architecture to reduce broadcast cost.
Leong's DHT [LL04]	ring	1	n	—	It spreads membership information by passing tokens.
Kelips [GBL ⁺ 03]	$\sqrt{n}$ groups	2	$O(\sqrt{n})$	—	Each host monitors hosts in its group and 2 nodes from each foreign group.
EpiChord [LLD04]	ring	$O(\log n)$	$O(\log n)$	no	it collect as many neighbour nodes as possible through bypassing messages.
Symphony [MBRI03]	small world	$O(\frac{\log n \cdot \log \log n}{\log s})$	s	no	s is a constant assigned by user.
Accordion [LSMK05]	small world	$O(\frac{\log n \cdot \log \log n}{\log s})$	s	yes	s is decided by bandwidth budget.
Tambour	small world	$O(\frac{\log n \cdot \log \log n}{\log s})$	s	yes	s is decided by lookup failure rate.

Table 2.1: A comparison of various DHT networks.

Chapter 3

Design Challenges

Most existing DHTs are designed to work well under a certain range of deployment scenarios. For example, many $O(\log n)$ -hop DHTs like Chord have a relatively high lookup hopcount and latency, but they only consume a small amount of bandwidth to maintain their routing structures. This type of protocols is suitable for large networks with frequent churn. However, for a smaller and stabler network, $O(1)$ -hop protocols like OneHop have best lookup latency at the expense of communication costs. Unfortunately, the costs of such protocols might be so high that exceed the network capacity as the system grows larger or suffers from high rate of churn. As most DHTs could only fit in one end of the operating spectrum, they will either produce non-optimal lookup performance or risk overloading the network with communication overhead when the operating environment is drifting away from their original design assumption.

In order to operate in a wide range of different environment, a robust DHT system should not rely on a fixed structure that only work well under certain circumstances, and it has to have the ability to adjust itself to achieve good performance in an unexpected operating conditions. Generally speaking, when a DHT needs to have lower lookup latency, it should acquire as many routing entries as possible; when the DHT need to be robust against changing operating environment, it should keep a smaller routing table and control the overhead traffic. Thus, an ideal DHT design, which is robust and produces reasonable performance, ought to be able to adapt a node's routing table size on the fly

according to the deployment environment. Such DHT would work like an $O(1)$ -hop protocol when the bandwidth is plentiful relative to the network size and churn, and perform multi-hop lookups like $O(\log n)$ -hop protocols when the network is unstable or has limited bandwidth.

Considering that many current popular Internet applications started as a small project or sometimes even a personal hobby, then grow into a monster in short period of time, we want a DHT protocol work well in both end of the operating environment spectrum. The ideal DHT protocol should have a flexible routing table size, and it could be tuned to fit current network size and churn rate. It would be even better if it is self-adaptive, since most users are not experts on P2P networks, they have no idea how to set up a DHT to fit their application environment, and usually, they leave the default parameters of a P2P programs unchanged. There are even some cases in which users report incorrect information to P2P programs. Thus, a DHT should avoid letting user to tune the system.

The ideal DHT should also have a flexible routing structure, since this will open a door to various of optimization techniques, like proximity neighbour selection (PNS) and random routing (RR). Proximity neighbour selection is a strategy that chooses a node with shortest delay to forward a lookup. This technique could improve the lookup performance by a large margin, so that, a DHT is considered useless in practice if it does not support it. Random routing, which adds randomization in each routing selection, could make a protocol more resilience to heavy churn.

Besides a flexible routing structure, we also want the routing table maintenance cost remains at a reasonable level for any types of operating environments. Usually, routing table maintenance contains two parts — to learn new neighbours and to ping current neighbour periodically. Putting more bandwidth on either of them could produce better lookup performance, while lowering the performance requirement could save a protocol some maintenance cost. No matter where the balance point between bandwidth and performance a DHT would prefer, it should try its best to avoid extreme cases.

The ideal DHT design should be robust against the constant changes of

operating environments and can tune its performance when the network grows quickly or is unstable. To achieve such level of robustness, a DHT has to have bounded bandwidth overhead, and under no circumstance, its routing table maintenance traffic can exceed the link capacities. Most existing DHTs pay no effort to bound its bandwidth, and their maintenance traffic is decided by the number of lookups, the size of the network and the churn rate. In order to assure robustness, Tambour should take the bandwidth consumption into account, and it should have some mechanism to reduce the risk of flooding the network when the lookup load is heavy or the size of the network is large.

To understand the challenges of designing an adaptive DHT protocol like Tambour, we address the characters of DHT networks in the following sections.

3.1 Routing Structure

Similar to Chord [SMK⁺01], Tambour maps nodes to a ring-like circular address space by their randomly selected 128-bit identifiers. By sending probe packet periodically, each node maintains a list of successor nodes, whose IDs most closely follow the node on the ring, and a list of predecessors accordingly. With this simple routing structure, a lookup request is forwarded among adjacent nodes clockwise on the ring until it reaches the node for which the request is looking. Since in each step, the lookup request reduces the clockwise ID distance to the destination by at least one, this procedure will finish in $O(\log n)$ hops eventually.

Obviously, the simple ring routing structure is not efficient and scalable enough for practical use. Chord solves this problem by maintaining a “finger-table” with $O(\log n)$ entries to fold the ID distance by half at each hop, so that, it guarantees $O(\log n)$ -hop lookup performance. Besides Chord, Pastry [RD01], Tapestry [ZHS⁺04], and Kademlia [MM02] employ similar mechanisms to achieve good scalability. However, in these DHTs, only a few routing entries are eligible to forward a given lookup. This inflexibility in choosing next hop not only reduces the efficiency of proximity neighbour selection (PNS) [BSV03], but also does pose a robustness problem.

Figure 3.1: Kleinberg’s generalization of the Small World Phenomenon.

One of the major differences between Tambour and previous DHTs is its routing structure. The routing structure is based on an interesting theory called the Small World Phenomenon, which states that the average path length for social network of people is short. This phenomenon is also referred as Six Degrees of Separation in popular culture. Kleinberg generalized the theory on a two-dimensional plane and stated that if every node knows four adjacent neighbours and one random long-distance neighbour, as long as the distance to the long-distance neighbour follows a $\frac{1}{x}$ distribution, there exists an algorithm that could find a $O(\log^2 n)$ -hop path between any two nodes on average [Kle00] (See Figure 3.1).

Tambour uses this $\frac{1}{x}$ distribution to populate its routing table. Each Tambour node can have an arbitrary number of neighbours as long as the distances to those neighbours follow that $\frac{1}{x}$ distribution. Figure 3.2 illustrates how this distribution looks like, where the dark dots represent neighbours of the node in the bottom. In general, the further two nodes are from each other on the ring, the less likely that one chooses the other as a neighbour. This $\frac{1}{x}$ distribution creates a simpler and more flexible routing structure. It has been used by several previous DHTs such as Symphony [MBRI03] and Accordion [LSMK05]. Interestingly,

Figure 3.2: A $\frac{1}{x}$ distribution of neighbours in Tambour

Chord uses $\frac{1}{x}$ distribution as well unintentionally, since its “finger-table” maintains a constant number of routing entries for every ID-range $[2^{i-1}, 2^i)$, and this matches the fact that the number of entries required by $\frac{1}{x}$ distribution, $\int_{2^{i-1}}^{2^i} \frac{1}{x} dx = \log 2$, is a constant as well.

With this $\frac{1}{x}$ distribution, Tambour gives a node the flexibility to utilize a routing table of any size. When the network has limited bandwidth or is unstable, Tambour maintains a small routing table to achieve Chord-like $O(\log n)$ -hop lookup performance; when the network environment permits, Tambour can scaling its performance up to constant hops like $O(1)$ -hop protocols [BSV03, GBL⁺03, RB04, GLR04, LL04]. Another advantage of $\frac{1}{x}$ distribution is its flexibility to select routes. Since each routing entry near the destination on the address space is a potential route, a node has a large set of routing entries to choose from as next hop to avoid nodes with low availability or high latency.

The $\frac{1}{x}$ distribution provides a flexible and scalable routing structure for DHTs. Based on this model, Tambour employs several other optimization tech-

niques to fulfill its advantages in different operating environments.

If each node uses the small world distribution to choose neighbours and keeps in track of only one neighbour, the average lookup hop count will be $O(\log^2 n)$ [Kle00]. By increasing the number of neighbours, the average lookup hop count reduces accordingly since it becomes more likely to find a shortcut in each step. In the case where each node has s neighbours, Jinyang Li *et al.* show that it requires $O(\frac{\log n \cdot \log \log n}{\log s})$ hops for one lookup request on average based on Kleinberg's proof framework [LSMK05]; and Gurmeet Singh Manku *et al.* give a similar, however, slightly different analysis result $O(\frac{1}{s} \log^2 n)$ [MBRI03].

Unfortunately, both theoretical analysis are flawed. With Kleinberg's proof framework, it is not possible to prove a better result than $O(\log n)$ hops, that is, no matter how many neighbours known by each node, it takes at least $O(\log n)$ hops to reach the destination. This is not saying that it is not possible to improve the hop counts by increasing number of neighbours when $s > \log n$. It is just the Kleinberg's method could not show stronger proof in that case. Normally, however, each node knows more than $\log n$ nodes. In fact, almost all classic DHTs like Chord, Pastry and Tapestry maintain at least $\log n$ neighbours for each node. Thus, it is a conservative and more theoretically correct estimation to assume that the hop counts is $O(\log n)$.

The routing structure of Tambour is both flexible and scalable due to the small world distribution. Even if there is only one long distance neighbour besides the neighbours in successor and predecessor list, lookups can still be routed in $O(\log^2 n)$ on average, thus, it is scalable. It is also flexible since each node can be a contributing neighbour for any other nodes in their routing tables. From a node point of view, every other node is a valid routing entry and a potential helper to provider fewer lookup hops.

Based on Kleinberg's analysis, not all probability distribution could achieve this small hopcount as the $\frac{1}{x}$ distribution does. Actually, if there is only one random long distance neighbour as described by Kleinberg, using $\frac{1}{x}$ distribution is the only way to achieve the scalable $O(\log^2 n)$ -hop lookups, and other distributions will do worse. Intuitively, nodes can make good forwarding progress continuously in ID space step by step as the lookup message gets closer to its

destination, since nodes know more about the regions that are near to itself than far away. On the contrary, with other distributions like a uniform random distribution, a lookup could get closer quickly in the first few steps, but has to switch to adjacent neighbours to move slowly to the key at 1 unit of ID distance a time. If the distribution is too dense at the beginning, the lookup process will be too slow in the first few steps when the lookup message is still far from the destination.

More formally, we can show that, if each node has s neighbours which are uniformly distributed in ID space, then lookups require $O(\sqrt{\frac{n}{s}})$ hops on average. Suppose each node has a successor entry and s random neighbours. We can estimate how often a node has to use its successor entry to relay a lookup rather than use long distance neighbours. Let y be the current node that is $d(y, k)$ distance away from the key k in ID space. Suppose y' is the next hop, then the probability of y' at distance d away from y is:

$$Pr(d(y, y') = d) = \frac{s}{n} \left(1 - \frac{d}{n}\right)^{s-1} < \frac{s}{n} \quad (3.1)$$

Thus, the average distance between y and y' is less than

$$\sum_{d=1}^{d(y,k)} d \cdot \frac{s}{n} < \frac{s}{n} \cdot \frac{d(y, k)^2}{2} \quad (3.2)$$

If $d(y, y') = \frac{s \cdot d(y, k)}{2n} < 1$, y has to use its successor entry to relay lookup messages. Hence, a total of $\sqrt{\frac{2n}{s}}$ successor hops are used to forward lookups on average, which proves the statement that lookups require $O(\sqrt{\frac{n}{s}})$ hops on average if neighbours are uniformly distributed.

3.2 Routing Update

A Tambour node starts updating its routing table by collecting new routing entries and removing stale ones continuously after it joins the network. If there are plenty of bandwidth available, a node should spend more bandwidth on expanding its routing table. To maintain a large routing table, a node should acquire new routing entries and update its routing table more frequently. The

Figure 3.3: The density of ID distances between a node and its neighbours after 30,000 lookups for random keys which matches a $1/x$ distribution well.

simplest method to learn new neighbours is exploring a certain region actively. However, this is not efficient in term of bandwidth overhead, and a more efficient approach is to monitor passing-by lookup messages, and learn about new neighbours and the liveness of existing routing entries from those ordinary lookup traffic.

The new neighbours that are learned from lookup traffic does not necessarily follow the $\frac{1}{x}$ distribution. In those DHTs which use iterative lookups [SMK⁺01, LLD04], the node who initiates a lookup would talk directly to each hop along the lookup request. Suppose the keys are uniformly distributed in ID space, the querying will follow a uniform distribution as well rather than desired small world distribution. It is possible to only pick some of the entries to form a small world distribution, however, since this removes information from the routing table, as long as a DHT allows flexible and arbitrary-sized routing tables, this method could only reduce the lookup performance.

Tambour employs another approach, called recursive routing, with which each hop forwards a lookup request to its next hop directly rather than returns

to the initial host and let it make the decision like iterative routing. Since each lookup request is broken down to relatively short hops, intuitively, there are more short lookups in the system than long lookups, and a node is more likely to contact near-by neighbours. A Tambour node piggy-backs additional routing entries which either are close to its own ID or follow the small world distribution to help previous hop create a $\frac{1}{x}$ distributed routing table. If the original lookups follow a uniform distribution, the ID distances of all lookups including lookup relays have a small world distribution. As a result, even if nodes start with random neighbours, learning from recursive lookups leads to small world distributed routing tables eventually. This effect can be demonstrated with a simple experiment in a static network. In Figure 3.3, each node has 20 random neighbours at the beginning of the simulation, then it starts to learn new routing entries from random recursive lookups initiated by random nodes in the network. After 30,000 lookups messages, the resulting probability density function of routing tables matches the small world distribution.

A Tambour node collects the majority of its routing entries from lookup traffic passing by. Because the identifier interval of each lookup hop follows a $\frac{1}{x}$ distribution if lookup keys are uniformly distributed in the circular identifier space, the neighbours learned from lookup traffic will fit the small-world distribution as a result. However, some lookup keys might be more popular than others in practice, which breaks the assumption of uniformity, hence, Tambour explores actively for new neighbours according to the small-world distribution.

Instead of pinging neighbours periodically, Tambour forwards each lookup request to several neighbours simultaneously to reduce the possibility of having to recover from stale routing entries. Parallel lookups is more efficient than active neighbour exploration in term of bandwidth overhead. The reason is that the larger degrees of parallelism, the more opportunities of learning new neighbours and refreshing the status of current neighbours, and at the same time, it reduces the risk of waiting for timeouts. In other words, parallel lookup not only reduces the negative effect of timeouts on the overall latency, but also does maintain the freshness of routing tables.

With parallel lookup capability, a node could send multiple lookup requests

to several neighbours at the same time. Intuitively, the more neighbours a node have, the better lookup latency it will get. As it showed before, large number of neighbours comes at a price. Thus, it is interesting to know that whether or not it is worth to actively search for new neighbours when the node have already had many neighbours with low latency. Following theorems will give a clue when is the right time to discover new neighbours.

Theorem 1 *Suppose nodes are uniformly distributed on Earth and the median latency between each pair of nodes is t_0 . For two random nodes, the probability of the latency between them being shorter than x is $\frac{1}{2}(1 - \cos \frac{x\pi}{2t_0})$.*

Proof The latency between a pair of nodes is positively correlated with the geographical distance between them [GGG⁺03]. The reason behind this phenomenon is that latency is limited by the speed of electronic signal transmission, meanwhile the data packets on the Internet is usually routed with shortest path. Therefore, if the latency between a pair of nodes is x , then,

$$\frac{\frac{\pi}{2}r}{t_0} = \frac{d}{x}, \quad (3.3)$$

where r is the radius of Earth and d is the distance between the two nodes.

Suppose the angle between the two nodes is α , then Equation 3.3 can be written as,

$$\alpha = \frac{d}{r} = \frac{x\pi}{2t_0} \quad (3.4)$$

Thus, the probability of the latency shorter than x is,

$$\begin{aligned} & Pr(\text{latency} < x) \\ &= \frac{2\pi r h}{4\pi r^2} \\ &= \frac{r - r \cos \alpha}{2r} \\ &= \frac{1}{2} \left(1 - \cos \frac{x\pi}{2t_0}\right) \end{aligned} \quad (3.5)$$

□

A node should check whether it is worth to search for new neighbours with Theorem 1 before doing any probe. If current neighbours are good enough and

the probability of getting better neighbours is low, The node does not have to waste bandwidth on active discovery. This indicator is useful for deciding when to update neighbour globally, however, the more important application is to find out which ID regions of neighbours are worth to update. With a limited bandwidth budget, Theorem 1 can be used to improve the cost effective of the bandwidth.

Sometimes the median latency of the network or the latency to the other nodes is not available to a node, the predication can only be based on the knowledge of the current number of neighbours. The following theorem is suitable for such scenario.

Theorem 2 *Suppose nodes are uniformly distributed on Earth. The expected minimal distance between a node and k other random nodes is approximately equal to $\frac{r\sqrt{\pi}}{\sqrt{k+1}}$, where r is the radius of Earth.*

Proof From the proof of Theorem 1, we know that,

$$F(x) = Pr(d < x) = \frac{1}{2}\left(1 - \cos \frac{x}{r}\right) \quad (3.6)$$

So the probability of having at least one node in k nodes whose distance is smaller than d is,

$$F_k(x) = 1 - (1 - F(x))^k \quad (3.7)$$

By the definition, the mathematical expectation of the minimal distance is,

$$\begin{aligned} & E[\min(d)] \\ &= \int_0^{+\infty} x \cdot f_k(x) dx \\ &= \int_0^{\pi r} x \cdot F'_k(x) dx \\ &= x \cdot F_k(x) \Big|_0^{\pi r} - \int_0^{\pi r} F_k(x) dx \\ &= \pi r - \int_0^{\pi r} 1 - (1 - F(x))^k dx \\ &= \int_0^{\pi r} (1 - F(x))^k dx \\ &= \int_0^{\pi r} \frac{1}{2^k} \left(1 + \cos \frac{x}{r}\right)^k dx \end{aligned} \quad (3.8)$$

Let $\alpha = \frac{x}{r}$, then

$$\begin{aligned}
 & E[\min(d)] \\
 &= r \cdot \int_0^\pi \left(\frac{1 + \cos \alpha}{2} \right)^k d\alpha \\
 &= r \cdot \int_0^\pi \left(\frac{1 + 2(\cos \frac{\alpha}{2})^2 - 1}{2} \right)^k d\alpha \\
 &= r \cdot 2 \int_0^\pi \left(\cos \frac{\alpha}{2} \right)^k d\frac{\alpha}{2} \\
 &= r \cdot 2 \int_0^\pi \left(\sin \frac{\alpha}{2} \right)^{2 \cdot \frac{1}{2} - 1} \left(\cos \frac{\alpha}{2} \right)^{2(\frac{1}{2} + k) - 1} d\frac{\alpha}{2}
 \end{aligned} \tag{3.9}$$

Based on the following formula¹:

$$B(m+1, n+1) \equiv 2 \int_0^{\frac{\pi}{2}} (\sin \theta)^{2m+1} (\cos \theta)^{2n+1} d\theta \tag{3.10}$$

We have

$$\begin{aligned}
 & E[\min(d)] \\
 &= r \cdot B\left(\frac{1}{2}, \frac{1}{2} + k\right) \\
 &= r \sqrt{\pi} \frac{\Gamma(\frac{1}{2} + k)}{\Gamma(1 + k)} \\
 &\approx \frac{r \sqrt{\pi}}{\sqrt{k+1}}
 \end{aligned} \tag{3.11}$$

□

Theorem 2 can be used to predict how many nodes a node has to probe before getting a better node as neighbour. With the consideration of bandwidth limit, a node should keep in mind that if it knows large number of neighbours in a certain ID region, the probability of discover a new neighbour with lower latency will not be high. It is more cost-effective to probe an ID region with relatively fewer known neighbours.

¹See <http://mathworld.wolfram.com/BetaFunction.html>

3.3 Stability of Node

Unlike most traditional distributed system studies, where systems are built among comparatively stable servers, the majority of the hosts in a peer-to-peer systems are just ordinary personal computers that have the Internet access. Understanding the pattern how long these unstable nodes will stay and contribute in a peer-to-peer system is a key factor to optimize the performance and provide better robustness for the system.

Many early studies on DHTs focused on how a system could survive large-scale failures. For example, several papers have shown that, theoretically, a DHT with a logarithmic number of neighbours per node can survive with high probability even when half of the nodes in the system fail simultaneously [SMK⁺01, KK03]. However, it has been shown in many recent research studies that, beside of large-scale simultaneous failures, DHTs and many other type of peer-to-peer systems also suffer from a steady and continuous process of nodes joining and leaving the system, a process called churn [GGG⁺03, RGRK04].

There are mainly two metrics of churn used in existing studies. A node's *session time* or *session duration* is the elapsed time between when it joins the network and when it subsequently leaves. the median session time has been observed to be within the range from as short as a few minutes to as long as an hour depending on the applications for which those systems were deployed [BSV03, CLL02, GDS⁺03, SGG02, SGG03, ZDL07].

In contrast, a node's *lifetime* is the time between when it enters the network for the first time and when it leaves the network permanently. The sum of a node's session times divided by its lifetime is often called its *availability*. While observed median session times are on the order of tens of minutes, median lifetimes are usually on the order of days, and median availability of around 30% [BSV03].

Some studies, especially the ones focused on data maintenance protocols, may use the combination of lifetime and availability as the metrics of churn, since it could help reduce the expense of data migration in theory [BR03, RB04]. However, with respect to membership management, or lookup functionality, of a DHT, the session time of a node is definitely a more important metric of

Figure 3.4: The cumulative distribution of node session duration in Gnutella network compared with an exponential distribution a Pareto distribution.

these two. Because even temporary loss of a routing neighbour weakens the correctness and performance guarantees of a DHT. If a node goes unavailable frequently, it would not be a good choice as a neighbour for other nodes and have little value in performing lookup relays.

Forwarding a lookup message to a neighbour that is no longer available does not only waste network bandwidth but also increases the latency as nodes need to wait multiple round trip times to declare the lookup message failed before trying another neighbour [CCR04, RGRK04]. In order to reduce the expenses of waiting for dead nodes during lookup relays, most existing DHTs contact each neighbour periodically to determine whether the routing entry is still valid [CCR04, DLS⁺04, MM02, ZHS⁺04]. By evicting neighbours that do not reply the probing messages for some interval from routing table, a node can make sure that its neighbours will be available with high probability when a lookup relay is coming, since they have been tested not long ago.

The probability distribution of node session time is similar to a Pareto distribution (See Figure 3.4, which is the cumulative distribution of node session

duration in Gnutella network [SGG03] compared with an exponential distribution using $\lambda = 0.015$ and a Pareto distribution using $\alpha = 0.7$ and $\beta = 17.1$). Tambour estimates a routing entry's liveness probability based on Δt_{alive} (the time between when the neighbour last joined the network and when it was last heard) and Δt_{since} (the time between when it was last heard and now). A routing entry's Δt_{alive} liveness probability (p) is approximated as $p \approx \frac{\Delta t_{alive}}{\Delta t_{since} + \Delta t_{alive}}$. The estimate is exact alive when the node lifetime follows a Pareto distribution with shape parameter $\alpha = 1$.

As the underlying routing structure gives the flexibility to pick next hop from a large pool, Tambour enhances the lookup performance by preferring routing candidates that are more responsive and stabler.

In order to assure the liveness of neighbour nodes and calculate their latencies, most exiting DHTs send probing message to each routing entry periodically [SMK⁺01, RD01, ZHS⁺04]. One of the weaknesses of this approach is that it limits the size of routing table by the available bandwidth and reduces the effect of proximity routing [BSV03]. Tambour employs a more efficient technique, parallel lookup, to avoid the probing overhead and mask timeouts caused by stale entries at the same time [LSM⁺05].

Without periodical contact, Tambour cannot be certain whether a neighbour is still in the network before relaying a lookup request to the node. Thus, Tambour assumes that the node lifetimes follow a heavy-tailed Pareto distribution as suggested by empirical studies [SGG03, ZDL07], and predicts the probability of a neighbour being alive with the knowledge about when the node joined the network and when it was seen last time [LSMK05]. Another problem caused by the lack of ping is that a node has no idea about the latency of a new neighbour right after learning it from other nodes. Tambour compensates this by bringing in the Vivaldi coordinate system, which predicts latencies among nodes at a small bandwidth cost [DCKM04]. After the first lookup request between two nodes, Tambour will update the latency data and Vivaldi system with firsthand knowledge.

While many DHTs would consider both the latency and liveness of a neighbour in routing selection, they usually give priority to one trait over the other.

Tambour characterizes node state in a more balanced way with the mathematical expectation of lookup latency. For example, if a neighbour with 80% chance of being alive could receive the lookup request in 150 milliseconds, it would also possibly lost the request in 20% of the cases and waste 1 second (5 times of the average Internet host latency in a typical setting) for the sender to recover, therefore, in average, the latency of routing through that neighbour is $150 * 80\% + 1000 * 20\% = 320$ milliseconds. In general, the expected lookup latency $E[t]$ is $t_{succ}p + t_{fail}(1 - p)$ where t_{succ}, t_{fail} are the latency and timeout respectively, and p is the probability of the neighbour being alive. Not only the mathematical expectation of lookup latency gives an all-around understanding of node state, but also does make it easier for Tambour to provide controllable average latency.

Chapter 4

The Tambour Protocol

One of the challenge every peer-to-peer system has to cope with is churn: nodes continuously join and leave the system. Studies of file sharing networks observe that the median time a node stays in the system ranges from tens of minutes to an hour depending on variant applications [GGG⁺03, YTD06, ZDL07]. It is not a good assumption that departing nodes will be able to notify their neighbours before leaving, and in many DHTs, nodes do not even know who have them as neighbours. Stale entries result in expensive lookup timeouts, since it takes multiple round-trip time for a node to determine that a lookup packet has been lost and re-route it through another neighbour. There are various methods to reduce lookup latency and increase the accuracy of routing tables under churn. In general, these methods generate extra communication to get more information about the liveness of existing neighbours and new nodes in the network. In order to be robust in scenarios when the network becomes unstable and churn rate increases rapidly, DHT networks should keep the extra cost in control to avoid flooding the network.

Tambour reduces average lookup latency by using parallel lookup and bounds the induced communication overhead automatically. While many popular DHTs organize neighbours in structured manners with a fixed routing table size. Tambour dynamically tunes its table size to get the best lookup performance. It does not use periodical probe to test the availability for most of its neighbours, but maintains a large routing table and selects next hop node in an opportunist

way. Most existing DHTs more or less rely on users to set up various parameters in order to tune the performance of the networks under different environments. Unfortunately, studies show that many of these parameters are in practice either left unspecified or deliberately misreported [SGG02]. Instead of asking user to do optimization, Tambour automatically provides the best lookup performance in a robust way and fits itself as the deployment environment changes. Performance evaluations show that Tambour keeps its communication overhead within a range over different operating conditions, and has similar or better lookup performance than some existing DHT protocols even when they were tuned for specific workload.

In this Chapter, we present in detail how a Tambour node organizes its routing table and acquires new routing entries via active exploration and learning from passing-by lookup requests.

4.1 Routing Structure

Similar to Chord [SMK⁺01], Tambour maps nodes to a ring-like circular address space by their randomly selected 128-bit identifiers. By sending probe packet periodically, each node maintains a list of successor nodes, whose IDs most closely follow the node on the ring, and a list of predecessors accordingly. With this simple routing structure, a lookup request is forwarded among adjacent nodes clockwise on the ring until reaches the node for which the request is looking. Since in each step, the lookup request reduces the clockwise ID distance to the destination by at least one, this procedure will finish in $O(\log n)$ hops eventually.

Obviously, the simple ring routing structure is not efficient and scalable enough for practical use. Chord solves this problem by maintaining a “finger-table” with $O(\log n)$ entries to fold the ID distance by half at each hop, so that, it guarantees $O(\log n)$ -hop lookup performance. Besides Chord, Pastry [RD01], Tapestry [ZHS⁺04], and Kademlia [MM02] employ similar mechanisms to achieve good scalability. However, in these DHTs, only a few routing entries are eligible to forward a given lookup. This inflexibility in choosing next hop

not only reduces the efficiency of proximity neighbour selection (PNS) [BSV03], but also does pose a robustness problem.

For a Tambour node, the ID distance from the node to each of its neighbours follows a $\frac{1}{x}$ distribution. In other words, the further two nodes are from each other on the ring, the less likely that one chooses the other as a neighbour. This $\frac{1}{x}$ distribution, which is based on the “small-world” model [Kle00], creates a simpler and more flexible routing structure. It has been used by several previous DHTs such as Symphony [MBRI03] and Accordion [LSMK05]. Interestingly, Chord uses $\frac{1}{x}$ distribution as well unintentionally, since its “finger-table” maintains a constant number of routing entries for every ID-range $[2^{i-1}, 2^i)$, and this matches the fact that the number of entries required by $\frac{1}{x}$ distribution, $\int_{2^{i-1}}^{2^i} \frac{1}{x} dx = \log 2$, is a constant as well.

With this $\frac{1}{x}$ distribution, Tambour gives a node the flexibility to utilize a routing table of any size. When the network has limited bandwidth or is unstable, Tambour maintains a small routing table to achieve Chord-like $O(\log n)$ -hop lookup performance; when the network environment permits, Tambour can scaling its performance up to constant hops like $O(1)$ -hop protocols [BSV03, GBL⁺03, RB04, GLR04, LL04]. Another advantage of $\frac{1}{x}$ distribution is its flexibility to select routes. Since each routing entry near the destination in the address space is a potential route, a node has a large set of routing entries to choose from as next hop to avoid nodes with low availability or high latency.

The $\frac{1}{x}$ distribution provides a flexible and scalable routing structure for DHTs. Based on this model, Tambour employs several other optimization techniques to fulfill its advantages in different operating environments.

4.2 Node State

As the underlying routing structure gives the flexibility to pick next hop from a large pool, Tambour enhances the lookup performance by preferring routing candidates that are more responsive and stabler.

In order to assure the liveness of neighbour nodes and calculate their latencies, most exiting DHTs send probing message to each routing entry periodically

[SMK⁺01, RD01, ZHS⁺04]. One of the weaknesses of this approach is that it limits the size of routing table by the available bandwidth and reduces the effect of proximity routing [BSV03]. Tambour employs a more efficient technique, parallel lookup, to avoid the probing overhead and mask timeouts caused by stale entries at the same time [LSM⁺05].

Without periodical contact, Tambour cannot be certain whether a neighbour is still in the network before relaying a lookup request to the node. Thus, Tambour assumes that the node lifetimes follow a heavy-tailed Pareto distribution as suggested by empirical studies [SGG03, ZDL07], and predicts the probability of a neighbour being alive with the knowledge about when the node joined the network and when it was seen last time [LSMK05]. Another problem caused by the lack of ping is that a node has no idea about the latency of a new neighbour right after learning it from other nodes. Tambour compensates this by bringing in the Vivaldi coordinate system, which predicts latencies among nodes at a small bandwidth cost [DCKM04]. After the first lookup request between two nodes, Tambour will update the latency data and Vivaldi system with firsthand knowledge.

While many DHTs would consider both the latency and liveness of a neighbour in routing selection, they usually give priority to one trait over the other. Tambour characterizes node state in a more balanced way with the mathematical expectation of lookup latency. For example, if a neighbour with 80% chance of being alive could receive the lookup request in 150 milliseconds, it would also possibly lost the request in 20% of the cases and waste 1 second (5 times of the average Internet host latency in a typical setting) for the sender to recover, therefore, in average, the latency of routing through that neighbour is $150 * 80\% + 1000 * 20\% = 320$ milliseconds. In general, if a node relays a lookup to d neighbours in parallel, assume that the latencies and live probabilities are $t_1, t_2, \dots, t_d$ and $p_1, p_2, \dots, p_d$ respectively, where $t_1 < t_2 < \dots < t_d$, then the mathematical expectation of the latency of this d degree parallel lookup is:

$$E[t] = \sum_{i=1}^d \left[t_i p_i \cdot \prod_{j=1}^{i-1} (1 - p_j) \right] + t_{fail} \cdot \prod_{j=1}^d (1 - p_j) \quad (4.1)$$

where t_{fail} is the timeout value. Not only the mathematical expectation of

Figure 4.1: A host forwards multiple lookups to neighbours near the destination key

lookup latency gives an all-around understanding of node state, but also does make it easier for Tambour to control the average latency.

4.2.1 Parallel Lookups

As illustrated in Figure 4.1, instead of pinging neighbours periodically, Tambour forwards each lookup request to several neighbours near the destination region simultaneously to reduce the possibility of having to recover from stale routing entries. It employs a lottery algorithm for neighbour selection, where each entry near the lookup destination on the ID ring is given a number of lottery tickets that is inversely proportional to the expected latency of that entry. This algorithm biases towards nodes with high availability and low latency, but nodes that are not so stable or responsive get some opportunity to be selected as well. With this randomized method, Tambour keeps picking neighbours one by one until the probability of the lookup being successfully received by at least one neighbour reaches a threshold q . In other words, if the average liveness probability is p , Tambour will create a d -way parallel lookup to insure that $1 - (1 - p)^d \geq q$.

It is nice to be able to guarantee the lookup delivery rate, since by doing so, Tambour would waste little time on waiting for the expensive timeout. However, this feature comes with a potential problem — if each node adopts a large degree of lookup parallelism arbitrarily to meet the delivery rate threshold, one lookup could trigger a flood of parallel messages in the whole system. As this problem affects the efficiency and robustness of Tambour, it is necessary to investigate whether it could happen on a regular basis.

The reason a node choosing a number of neighbours to do parallel lookup is because of the poor stability of these neighbours. In order to maintain the delivery guarantee, it has to use the degree of parallelism to compensate the probability of liveness. Once the node gets acknowledgements from some of the neighbours, it will increase the estimations of their likelihoods of being alive and remove those neighbours that never reply from the routing table. Therefore, the average liveness probability of the routing table is increased by the process, which also reduces the requirement of parallel degree for future lookups. Intuitively, this “refresh” effect neutralizes the risk of continuous whole-system flooding.

In fact, when the average liveness probability reaches an equilibrium between the “refresh” effect, which increases the liveness of neighbours with lookup feedback, and the “ageing” effect, which decreases the liveness of neighbours with no recent contact, the liveness probability p will have limited impact on overall cost of parallel lookups. If $p > q$, this statement is obviously true since no parallel lookup is required; if $p \leq q$, the equilibrium of p implies:

$$p = \left(\frac{s - dk}{s} \cdot p + \frac{dk}{s} \cdot 100\% \right) \cdot (1 - r) \quad (4.2)$$

where s is the size of routing table, k is the number of lookup requests in a unit of time, and r is node failure rate. Then, the cost parallel lookups is:

$$dk = \frac{sr}{1 - r} \cdot \frac{p}{1 - p} \leq \frac{sr}{1 - r} \cdot \frac{1}{1 - q} \quad (4.3)$$

This equation shows that, no matter how much the probability p is, the cost dk is always bounded by other factors, such as, the size of routing table and churn rate. Therefore, even with a low level of liveness probability, the guarantee of lookup delivery rate in Tambour will not overload the system in long term.

Equation 4.3 also implies that a Tambour node has to maintain a smaller routing table under heavy churn to constrain the control overhead. Since churn leads to evictions of neighbours and decreases the size of the routing table, there is a natural tendency which limits the overhead of parallel lookups. To make this process go smoother and avoid massive amount of lookups under churn, Tambour gives higher priority to nodes with low expected latency when node failure rate increases. It does not do so by adding more lottery tickets to more

reliable nodes but by increasing the size of the candidate pool. For example, when the churn is low, Tambour might only pick next hop from 8 nodes near the destination; when the network becomes more unstable, Tambour would pick from 16 candidates. With a larger candidate pool, stabler neighbours get higher probability to be picked, and neighbours with low probability of being alive have little chance to be “refreshed” and will be removed from the routing table through the evicting process eventually. This method has the similar effect as removing unstable nodes from routing table directly. Its advantage is that, if the high level of churn ends in a short period of time, it will not lose much information about those unstable nodes, which helps the system recover from the churn faster.

4.2.2 Routing Table Maintenance

A Tambour node collects the majority of its routing entries from lookup traffic passing by. Because the identifier interval of each lookup hop follows a $\frac{1}{x}$ distribution if lookup keys are uniformly distributed in the circular identifier space [LSMK05], the neighbours learned from lookup traffic will fit the small-world distribution as a result. However, some lookup keys might be more popular than others in practice, which breaks the assumption of uniformity, hence, Tambour adopts two methods to correct the routing entry distribution.

First, a Tambour node piggybacks several routing entries, which follow a small-world distribution from the recipient’s point of view, in each lookup and acknowledgement. And, similar to the lottery algorithm used in parallel lookups, the extra routing entries are randomly picked with priority to stable and low-latency neighbours. Second, Tambour explores actively for new neighbours according to the small-world distribution. For every time interval, a Tambour node asks a neighbour for routing entries in the ID range between that neighbour and the very next neighbour in the circular ID space. Since the neighbour is closer to the range than the current node, it knows more routing entries in the range. With this extra knowledge, the neighbour answers with the nodes which have lowest latency to the sender according to Vivaldi coordinate system.

The selection of which ID range to explore is based on whether it needs new neighbours in that range to match the small-world distribution or how much the latency could be improved with new neighbours. According to the small-world distribution, the number of nodes between neighbours i and $i + 1$ should be proportional to $\int_{a_i}^{a_{i+1}} \frac{1}{x} dx = \log \frac{a_{i+1}}{a_i}$, where a_i, a_{i+1} are the ID distances from current node to neighbour i and neighbour $i + 1$ respectively. On the other hand, the level of latency improvement with new neighbours is characterized by the ratio of the current expected lookup latency $E[t_i]$ for the range to the expected latency $E[t'_i]$ of new neighbours. By the nature of Tambour's parallel lookup algorithm, $E[t_i]$ is the expected latency of the several neighbours near the ID range forwarding lookup in parallel, and its value can be calculated from Equation 4.1. Since nodes reply exploration requests with the low-latency entries based on their best knowledge, $E[t'_i]$ is approximately equal to the lowest latency of nodes in that range. Generally, the more nodes are in a set, the shorter latency can be found. In this case, where the system is supposed to be deployed on the surface area of Earth, the lowest latency is approximately proportional to $(a_{i+1} - a_i)^{-\frac{1}{2}}$ [ZGG03], and so is $E[t'_i]$.

Combining above analysis, Tambour selects an ID range to explore randomly with a lottery algorithm which assigns the i th range a number of lottery tickets with following formula:

$$E[t_i] \cdot \sqrt{a_{i+1} - a_i} \cdot \log \frac{a_{i+1}}{a_i} \quad (4.4)$$

This exploration process approximates a $\frac{1}{x}$ distribution with the consideration of locality. It keeps hop-count low while providing good latency level for each hop to any part of the ID space.

Chapter 5

Evaluation

In this Chapter, we present the evaluation results of a prototype Tambour implementation and compare it to existing DHTs in simulation environments. The evaluation demonstrates our basic design concept that is to make Tambour tune itself to balance the tradeoffs between bandwidth overhead and lookup latency automatically. Tambour shows better performance than existing DHT protocols over a wide range of workloads and various churn rate, and it could bound its bandwidth consumption without any user intervention. When the churn is heavy, Tambour nodes decrease the sizes of their routing tables and increase the number of hops to match the performance of common $O(\log n)$ -hop protocols. On the contrary, when the network environment is stable and bandwidth is abundant, nodes expand their routing tables gradually and match or even exceed the performance of $O(1)$ -hop protocols eventually.

5.1 Experimental Setup

The prototype Tambour is implemented in a generic peer-to-peer protocol simulator, p2psim [p2p09]. We compare it with some other DHT implementations in p2psim, such as, Chord [SMK⁺01], OneHop [LL04] and Accordion [LSMK05], and investigate various characteristics of Tambour with simulations. The p2psim simulator is written in C++, and it can run experiments faster than most other available simulators. Thus, we can test Tambour's performance in

Figure 5.1: Tambour’s latency vs. bandwidth trade off compared to Chord and OneHop, using a 3000-node network and a churn.

many different network environments with various churn rate and workload settings. Since Chord shows best lookup performance with little bandwidth consumption when the network is large and OneHop is the best DHT with large bandwidth consumption when the network is small. Therefore, we focus our estimation of Tambour based its comparison with Chord and OneHop in simulation environments.

The experiments run on a network of 10,000 nodes. Each node is assigned a coordinator on a 2-dimensional plain, and the round-trip time between each node pair is proportional to their distance on the plain. The average round-trip time is about 170 ms. The node lifetimes follow a Pareto distribution with a mean of 60 minutes ($\alpha = 0.83$ and $\beta = 1560$ sec). Nodes that depart will rejoin under different addresses after an exponentially-distributed interval with a mean of 6 minutes.

Figure 5.2: The average routing table size for Chord and Tambour as a function of the average, using a 3000-node network and a churn.

5.2 Latency and Bandwidth

Tambour is designed to be self-adaptive to different operating environments and churn rates to achieve best possible performance. Figure 5.1 plots the trade-offs between average lookup latency and bandwidth overhead in OneHop, Chord and Tambour. In this experiment, Tambour gets the varieties of bandwidth overhead levels from different exploration rate settings. Tambour could achieve better average lookup latency than both Chord and OneHop no matter how much bandwidth they use. Especially, when the bandwidth is large, Tambour performs better than OneHop in term of average lookup latency, and when the bandwidth is limited, Tambour produces similar performance as Chord. This is due to Tambour's flexible routing table and efficient neighbour exploration algorithm.

When the bandwidth cost is small as showed in figure 5.2, Tambour maintains a small routing table similar as Chord's. With larger bandwidth cost, Tambour increases its routing table gradually to the size of the system, i.e., the number of live nodes in the system. As both protocols consume more band-

width, Chord increases the size of its routing table as well, however, it is not as quickly as Tambour because of its rigid routing structure, even Chord is optimized to produce best performance. Chord spends most of the extra bandwidth on routing entry maintenance rather than discovering new neighbours and growing its routing table. Comparing to Chord, Tambour increases its routing table faster when bandwidth increases. As a result, Tambour can have a large routing table like OneHop and reduce the average lookup latency by decreasing the number of hops per lookup.

Unlike Chord and Tambour, OneHop always keeps the complete status of the network on every single node in the network without exception. All events of node joining and leaving the work have to be propagated to other nodes in the system. This design consumes a lot of bandwidth, especially in a comparatively large network. Thus, there is no way we can configure OneHop to match Chord and Tambour at low bandwidth cost. If the bandwidth overhead for OneHop is not sufficient, joining and departing event will be propagated much slower, and the routing tables of OneHop will be less up-to-date, resulting in more timeouts and longer delays. Tambour does not update routing entries and compensates the risk of timeouts by using parallel lookups. As a result, Tambour is not as sensitive to the bandwidth as OneHop does, and it could employ a much smaller routing table under heavy churn or in a large system. Thus, Tambour can have an arbitrary large routing table without the penalty of lookup timeouts and routing entry maintenance cost.

5.3 Lookup Workload

In the previous section, we demonstrated the simulation results of Tambour and other DHTs in churn intensive environments where the churn rate is heavy relative to the lookup workloads. This section simulates Tambour under a heavy lookup workloads. Comparing to previous simulations in which each node initiates one lookup request every 4 minutes, in this experiment, each node starts a lookup every 9 seconds, and the churn rate is at the same level as previous section.

Figure 5.3: Tambour's lookup latency vs. bandwidth overhead tradeoff compared to OneHop and Chord.

Figure 5.3 shows the simulation results for Chord, OneHop and Tambour. In this lookup intensive operating environment, all of the three produces similar level of performance, while Tambour and OneHop are slightly better in term of lookup latency. Again, Tambour and Chord outperform OneHop in low bandwidth settings. In the churn intensive workload, both Tambour and Chord can operate with little bandwidth, however, in this experiment where lookup loads are much heavier, Tambour shows better adaptation to low level of bandwidth consumption than Chord. In a low lookup intensive environment like in Figure 5.1, Chord uses a small routing table and more lookup hops to lower the bandwidth consumption. However, this strategy does not work as well relatively with a high lookup rate like in Figure 5.3. High lookup rate brings much useful information about the network and neighbour nodes, however, Chord cannot take advantage of the extra information because of its rigid routing table structure. On the contrary, Tambour learns as many new routing entries as possible from lookup traffic, thus, intensive lookup workloads lead to a larger routing table, and as a result, Tambour needs fewer hops and less lookup overhead to do the hops. Therefore, Tambour could operate at the low bandwidth levels

Figure 5.4: The bandwidth overhead as a function of the lookup load for Chord, Accordion and Tambour.

where Chord could not work anymore.

Figure 5.4 plots the bandwidth costs of Chord, Accordion and Tambour as the lookup load increases. Since Chord does not use parallel lookup, its bandwidth consumption increases linearly with the lookup load. Accordion holds its bandwidth consumption at a steady level due to its sophisticated bandwidth budget system. Although Tambour has no direct control over its bandwidth, its parallel lookup algorithm scales well with increasing lookup load. Actually, the extra lookup load brings more routing entries and refreshes neighbour states, as a result, Tambour needs fewer hops, smaller degrees of parallelism and smaller extra overhead.

5.4 Churn Rate

Previous sections shows Tambour's ability to adapt to different lookup workloads; this section evaluates how it works with different levels of churn rates.

Figure 5.5 plots the lookup latency and bandwidth overhead of Chord, One-Hop and Tambour as a function of median node lifetime. The general trend of

Figure 5.5: The average lookup latency of Tambour, Chord and OneHop as a function of median node lifetime.

this figure is that, as the median node lifetime increases, the network becomes more stable and the average lookup latency turns lower and lower. Tambour's lower average lookup latency with longer median node lifetimes is because of that Tambour maintains a larger routing table and evicts fewer nodes in a unit of time when the network is comparatively stable. Chord's lower average lookup latency with longer median node lifetimes is due to that it has lower risk of forwarding lookup to a stale neighbour when the network is stable. Tambour is more consistent than both Chord and OneHop in term of lookup latency under churn. OneHop does not work well under high churn since it requires much more bandwidth to maintain its routing table. When the bandwidth is limited and the network is unstable, there will be many stale neighbours in OneHop's routing tables, and as a result lookup timeouts happen frequently due to the dead routing table entries.

Figure 5.5 shows the bandwidth consumption of Chord, OneHop and Tambour in the same experiment as previous figure. Although Tambour has no direct control over its bandwidth, its parallel lookup algorithm scales well under different churn rates. While Tambour could be able to maintain a relatively

Figure 5.6: The average bandwidth consumption by Tambour, Chord and OneHop as a function of median node lifetime.

fixed communication overhead no matter the churn level is, the bandwidth costs of both Chord and OneHop increase when the network is under heavy churn.

Chapter 6

Conclusion and Future Work

In order to avoid overloading the network with excessive control messages, a distributed system should be conscious of the control communication overhead under rapid system growth. Distributed hash tables, which are designed to be the building block of various peer-to-peer applications over the Internet, have to consider their feasibility of operating in environments range from few hundreds of nodes to millions of nodes.

This thesis has shown a new scalable and robust DHT protocol, Tambour, which bounds its bandwidth consumption according to a bandwidth budget. Under the bandwidth constrain, Tambour adjust a node's routing table size automatically based on churn rate. It uses parallel lookups to reduce lookup failures and latency, meanwhile it acquires new routing entries from the lookup process. Tambour estimates the probabilities of routing entries' liveness based on statistics of node lifetime history and evicts dead entries after lookup failures. When the network is unstable, or say, has larger churn rate, more routing entries will be evicted in a given period of time, and the routing tables will be getting smaller which minimize the number of timeouts for later lookup requests. Thus, Tambour's final routing table size is the equilibrium between the learning and evicting processes.

Performance evaluations show that Tambour keeps its communication overhead within the budget over a wide range of operating conditions, and has similar or better lookup performance than some existing DHT protocols even

when they were tuned for specific workload. When bandwidth is plentiful, Tambour provides lookup latencies and maintenance overhead similar to that of OneHop. When bandwidth is scarce, Tambour has lower lookup latency and less maintenance overhead than Chord.

A self-tuning, bandwidth-efficient protocol such as Tambour has several distinct advantages. First, Tambour controls and bounds its communication overhead to avoid overloading the network. Thus, Tambour is robust across different operating environments, especially in scenarios when the system faces rapid growth or the churn surges. Second, Tambour employs a node liveness probability framework to reduce the risks of losing all parallel lookup messages at the same time with minimum cost. Third, Tambour does not need manual parameter tuning at all. Users often do not have the expertise to tune each DHT parameter correctly for a given operating environment. Tambour shifts the burden of tuning from the user to the system by automatically adapting itself to the observed churn and workload to provide the best performance.

While the simulation results of Tambour on *p2psim* are promising, we have yet had the opportunity to do experiment in more realistic environment. As a part of the future work, we plan to implement Tambour on large-scale network testbed such as PlanetLab [Pla09], and deploy Tambour in some real world peer-to-peer application. There is a number of improvements that might affect Tambour's performance or usability in real world deployment.

For future study, it is interesting to see the feasibility of fusing Tambour into other existing DHT systems. Due to its flexibility and simplicity, a Tambour node should be able to work with DHTs like Chord and Accordion after a little modification.

The IP address of a node on the Internet could provide a lot of useful information about the node. For example, many Internet WHOIS databases have the nation and timezone data of any given IP address. With the timezone information, it is easy to calculate the local time of the location where a node is. If a node's local time is just near the end of office hour, e.g. 5pm, it has higher probability to be turned off at that time than normal, thus, it is a better idea to find another node as next hop. A node's Internet service provider (ISP)

and geographic coordination information, which are extremely helpful, can be found on WHOIS databases as well. Current Tambour implementation uses Vivaldi coordinates to predict latency between two nodes [DCKM04]. With ISP and geographic information, Tambour could make the latency prediction more accurate and achieve better lookup performance as a result.

Bibliography

- [BAS04] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 353–366, New York, NY, USA, 2004. ACM Press.
- [Bit09] <http://www.bittorrent.com/>, March 2009.
- [BR03] Charles Blake and Rodrigo Rodrigues. High availability, scalable storage, dynamic peer networks: pick two. In *HOTOS'03: Proceedings of the 9th conference on Hot Topics in Operating Systems*, pages 1–6, Berkeley, CA, USA, 2003. USENIX Association.
- [BSV03] Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker. Understanding availability. In *IPTPS*, pages 256–267, 2003.
- [CCR04] Miguel Castro, Manuel Costa, and Antony Rowstron. Performance and dependability of structured peer-to-peer overlays. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, pages 9–18, Washington, DC, USA, 2004. IEEE Computer Society.
- [CLL02] Jacky Chu, Kevin Labonte, and Brian Neil Levine. Availability and locality measurements of peer-to-peer file systems. In *In Proceedings of ITCOM: Scalability and Traffic Control in IP Networks*, pages 310–321, 2002.
- [DCKM04] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. In *In SIGCOMM*, pages 15–26, 2004.

- [DLS⁺04] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. Designing a DHT for low latency and high throughput. In *in NSDI*, pages 85–98, 2004.
- [DZD⁺03] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiawicz, and Ion Stoica. Towards a common API for structured peer-to-peer overlays. *Peer-to-Peer Systems II*, pages 33–44, 2003.
- [eDo09] <http://www.edonkey2000.com>, March 2009.
- [eMu09] <http://www.emule-project.net/>, March 2009.
- [Fas09] <http://developer.berlios.de/projects/gift-fasttrack/>, March 2009.
- [FFM04] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with Coral. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI '04)*, pages 239–252, San Francisco, California, March 2004.
- [Fre09] <http://freenetproject.org/>, March 2009.
- [GBL⁺03] Idnranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robert van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, pages 160–169, Berkeley, CA, 2003.
- [GDS⁺03] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. *SIGOPS Oper. Syst. Rev.*, 37(5):314–329, 2003.
- [GGG⁺03] Krishna Gummadi, Ramakrishna Gummadi, Steve Gribble, Sylvia Ratnasamy, Ion Stoica, K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. pages 381–394. ACM Press, 2003.

- [GLR04] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues. Efficient routing for peer-to-peer overlays. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 113–126, Berkeley, CA, USA, 2004. USENIX Association.
- [Gnu09] <http://rfc-gnutella.sourceforge.net/>, March 2009.
- [Kaz09] <http://www.kazaa.com/>, March 2009.
- [KK03] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. pages 98–107, 2003.
- [Kle00] Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. In *in Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 163–170, 2000.
- [LL04] Ben Leong and Ji Li. Achieving one-hop DHT lookup and strong stabilization by passing tokens. In *Proceedings of the 12th International Conference on Networks (ICON)*, pages 344–350, November 2004.
- [LLD04] Ben Leong, Barbara Liskov, and Erik D. Demaine. EpiChord: Parallelizing the Chord lookup algorithm with reactive routing state management. In *12th International Conference on Networks (ICON)*, pages 1243–1259, Singapore, November 2004.
- [LNBK02] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of ACM PODC*, pages 233–242, Monterey, CA, July 2002.
- [LSG⁺04] Jinyang Li, Jeremy Stribling, Thomer M. Gil, Robert Morris, and M. Frans Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS04)*, pages 87–99, San Diego, CA, February 2004.
- [LSM⁺05] Jinyang Li, Jeremy Stribling, Robert Morris, M. Frans Kaashoek, and Thomer M. Gil. A performance vs. cost framework for evalu-

- ating DHT design tradeoffs under churn. In *Proceedings of the 24th Infocom*, pages 225–236, Miami, FL, 2005.
- [LSMK05] Jinyang Li, Jeremy Stribling, Robert Morris, and M. Frans Kaashoek. Bandwidth-efficient management of DHT routing tables. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 99–114, Berkeley, CA, USA, 2005. USENIX Association.
- [MBRI03] Gurmeet Singh Manku, Mayank Bawa, Prabhakar Raghavan, and Verity Inc. Symphony: Distributed hashing in a small world. In *In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, pages 127–140, 2003.
- [MM02] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *IPTPS*, pages 53–65, 2002.
- [MN05] James W. Mickens and Brian D. Noble. Predicting node availability in peer-to-peer networks. *SIGMETRICS Perform. Eval. Rev.*, 33(1):378–379, 2005.
- [MNR02] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the 2002 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 183–192, New York, NY, USA, August 2002. ACM Press.
- [Nap09] <http://www.napster.com>, March 09.
- [p2p09] P2Psim: A simulator for peer-to-peer (P2P) protocols, March 2009.
- [Pla09] <http://www.planet-lab.org>, March 2009.
- [RB04] Rodrigo Rodrigues and Charles Blake. When multi-hop peer-to-peer routing matters. In *3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 112–122, 2004.

- [RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.
- [RFH⁺01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172, San Diego, CA, August 2001.
- [RGK⁺05] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiawicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. OpenDHT: a public DHT service and its uses. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 73–84, New York, NY, USA, 2005. ACM.
- [RGRK04] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiawicz. Handling churn in a DHT. In *In Proceedings of the USENIX Annual Technical Conference*, pages 127–140, 2004.
- [SDRC82] J. F. Shoch, Y. K. Dalal, D. D. Redell, and R. C. Crane. Evolution of the ethernet local computer network. *Computer*, 15(8):10–27, 1982.
- [SGG02] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking Conference (MMCN)*, pages 156–170, San Jose, CA, USA, 2002.
- [SGG03] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *Multimedia Systems*, 9(2):170–184, August 2003.
- [Sky09a] <http://arstechnica.com/tech-policy/news/2009/03/skype-handles-more-international-calls-than-att.ars>, March 2009.
- [Sky09b] <http://www.skype.com>, March 2009.
- [SLC⁺06] Jeremy Stribling, Jinyang Li, Isaac G. Councill, M. Frans Kaashoek, and Robert Morris. OverCite: A distributed, cooperative CiteSeer.

- In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '06)*, pages 143–153, San Jose, CA, May 2006.
- [SMK⁺01] Ion Stoica, Robert Morris, David Karger, Frans F. Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, October 2001.
- [SMK08] Emil Sit, Robert Morris, and M. Frans Kaashoek. UsenetDHT: A low-overhead design for Usenet. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI '08)*, pages 133–146, San Francisco, CA, April 2008.
- [SMPD05] Daniel Sandler, Alan Mislove, Ansley Post, and Peter Druschel. FeedTree: Sharing web micronews with peer-to-peer event notification. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 141–151, 2005.
- [SW05] Ralf Steinmetz and Klaus Wehrle. *Peer-to-Peer Systems and Application*, volume 3485 of *Lecture Notes in Computer Science*. Springer, September 2005.
- [WWW09] <http://www.w3.org/>, March 2009.
- [YTD06] Zhi Yang, Jing Tian, and Yafei Dai. Towards a more accurate availability evaluation. In *IWNAS '06: Proceedings of the 2006 International Workshop on Networking, Architecture, and Storages*, pages 73–80, Washington, DC, USA, 2006. IEEE Computer Society.
- [ZDL07] Mo Zhou, Yafei Dai, and Xiaoming Li. A measurement study of the structured overlay network in P2P file-sharing systems. *Adv. MultiMedia*, 2007(1):10–18, 2007.
- [ZGG03] Hui Zhang, Ashish Goel, and Ramesh Govindan. Incrementally improving lookup latency in distributed hash table systems. In *ACM Sigmetrics*, pages 114–125, 2003.
- [ZHS⁺04] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient

BIBLIOGRAPHY

69

global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53, 2004.

- [ZRM02] Lidong Zhou, Robbert Renesse, and Michael Marsh. Implementing IPv6 as a peer-to-peer overlay network. In *In Workshop on Reliable Peer-to-Peer Distributed Systems, 21st IEEE Symposium on Reliable Distributed Systems (SRDS '02)*, pages 347–351, 2002.