ABSTRACT

WANG, CHIH-CHIANG. Managing Heterogeneity in Structured Peer-to-Peer Systems. (Under the direction of Assistant Professor Khaled Harfoush).

Peer-to-Peer (P2P) systems account for more than 60% of today's Internet traffic. P2P success is attributed to the virtually unlimited capacity offered by its members, the resilience to failures, and features such as anonymous service. P2P systems are inherently heterogeneous. Members may have different service capacities, different Internet connectivity speeds, and spend different times in the system. Heterogeneity impacts the scalability of P2P systems, its ability to serve user requests and the perceived user performance. P2P systems need to be engineered to accommodate node heterogeneity. In this dissertation, we present two studies on node heterogeneity in structured P2P systems.

In the first study, we analyze routing performance in the presence of P2P overlay irregularity caused by the heterogeneous capacity of peers. We pinpoint how Greedy routing deviates from shortest path routes and propose a novel routing protocol, RASTER, which uncovers shortest routes in irregular overlays. In the second study, we provide an analytical model that captures the tradeoff between the stability and the scalability in P2P systems. We use the model to demonstrate that P2P instability can be mitigated and system throughput can be optimized through careful selection of serving members and data redundancy parameters. Our model reveals appropriate trends in engineering P2P systems under bandwidth and storage constraints, the skew in the size and popularity of maintained content, and the workload imposed on the system. We highlight the performance limitations of P2P systems as expressed in terms of the system throughput and response time to user requests in the presence of node instability. Our results challenge common wisdom about P2P systems, their appropriateness for different applications, and their virtually unlimited capacity.

Managing Heterogeneity in Structured Peer-to-Peer Systems

by

Chih-Chiang Wang

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, NC

2007

Approved By:

Dr. Injong Rhee	Dr. Rudra Dutta
Dr. Khaled Harfoush Chair of Advisory Committee	Dr. George Rouskas

Dedication

To my loving parents, my sister, and my brother in law, for their encouragement and support over these many years.

Biography

Chih-Chiang Wang was born in Kaohsiung, Taiwan. He received his Bachelor of Science degree and Master of Science degree from the Department of Electrical and Computer engineering, North Carolina State University, Raleigh, NC, USA, in year 2000 and 2002, respectively. From year 2003 to 2007, he has been pursuing his Doctor of Philosophy degree in the Department of Computer Science, North Carolina State University. Chih-Chiang is interested in research of architecture, routing protocols, and algorithms for distributed systems/applications, overlay and sensor networks. His current interest lies in the theoretical and foundational aspects of scalable and highly available distributed information systems such as scalable publishing/subscribing systems, distributed storage systems and peer-to-peer content distribution.

Acknowledgements

I have to admit that I knew absolutely nothing about academic research before I met my Ph.D. advisor. Literally speaking, I did not know how to think, explain, and tackle research problems like a genuine researcher, and I even did not know how to write a descent technical paper. If I have had any achievement in my research area at all, one person, above all, is directly responsible for that - my advisor Dr. Khaled Harfoush. As his first Ph.D. student, I am very grateful to have had the most opportunities to work closely with him. Indeed, the amount of time and patience he spent on me is incredible. To me Dr. Harfoush is not only my advisor, but also a mentor showing me the way of being a front-line researcher and giving me feedback and suggestions that are valuable to my future career. I also thank him for giving me complete freedom to deal with my work in the style I was comfortable with. I shall leave school with a stash of memories of the time we spent together, especially memories of many occasions when I attempted to sell him my ingenious ideas.

I would like to express my gratitude to Dr. David Thuente, the Director of Graduate Programs, and to the secretaries of Graduate Programs of the Department of Computer Science. Even though school funding has been tight over the past several years, one semester after another Dr. Thuente has always tried his best to squeeze something out of his limited budget to take care of students enrolled in the graduate program, me included. I especially appreciate Dr. Thuente's patience and generosity at the occasion when we have missed the milestone required for completion of the degree. The secretaries of Graduate Programs are always there for us, patiently dealing with our endless troubles and questions. All I can say is that I can hardly ask for any more from a graduate program.

I also want to thank many faculty of the Department of Computer Science and the Department of Electrical and Computer engineering of North Carolina State University for their kind help and guidance over these years. Dr. George Rouskas, thank you for your insightful discussion and constructive feedback to my work, which led to my first INFOCOM paper; Dr. Michael Devetsikiotis, thank you for helping me enroll into the Ph.D. program when no one else seemed care; thanks also go to my committee members, Dr. Injong Rhee and Dr. Rudra Dutta.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This decade has witnessed an explosive growth in the use of *Peer-to-Peer* (P2P) systems to exchange information such as HTML, plain text, music and movie files over Internet. As opposed to the traditional *Client/Server* (C/S) architecture that relies on dedicated machines (servers) to serve information to end users (clients), nodes in P2P systems function as both clients and servers to each other. Thus, as more peers join in the system, they not only consume but also add to the aggregate capacity of the system; this is generally referred to as P2P's *self-scalability ability*.

Napster [4] is the first P2P file sharing system that started this trend of massive, large-scale information exchange. Launched by Shawn Fanning in June, 1999, Napster allows peers to share MP3 music files over the Internet under the control of a centralized directory server. Within a year, the user population of Napster had reached to a peak of over 50 Million users, making it the fastest growing Internet application ever in history. A great portion of Napster's success is attributed to its use of the centralized server to maintain only the location and availability of peers and their shared files while leaving the actual file transfer between the requester and the file owner. Thus, the Napster solution does not require much bandwidth from dedicated servers compared to the traditional C/S architecture. However, Napster stil has the flaw of a single point of failure because of its centralized search infrastructure. The lawsuit filed against Napster by the Recording Industry Association of America (RIAA) in December, 1999 also brings out legal issues to the use of such systems.

A new generation of P2P file sharing systems such as Gnutella and KaZaa [2, 3] emerged shortly after RIAA's lawsuit against Napster. Unlike Napster, these so called *unstructured P2P systems* connect all peers into an unstructured overlay network in which arbitrary peers *connect* with each other and search for files is done by propagating queries throughout the network. The main disadvantage of unstructured P2P systems is that queries may not always be resolved. Since unstructured P2P systems do not maintain a global correlation between peers and their shared files, peers are likely not to know which peers hold the requested files or whether such requested files do exist in the system. In the worst scenario, the queries for a rare file may be flooded all over the system, which imposes a significant amount of unnecessary traffic on the network. Thus while the centralized search infrastructure, such as Napster's, leaves the system vulnerable to attacks and lawsuits, unstructured P2P systems' flooding-based search is significantly less efficient and stresses the network resources.

In order to attain the best of both worlds, search efficiency and deterministic results, research efforts have targeted a different approach, *structured* P2P systems, which organize nodes in pre-defined overlay structures. By far the most adopted class of structured P2P systems rely on Distributed Hash Tables (DHTs), which were launched in 2001. DHTs such as CAN, Chord, Tapestry, and ODRI [43, 51, 56, 32] are decentralized distributed systems that offer data naming and data location through a distributed hash-table-like lookup service. Unlike unstructured P2P systems, DHTs employ a globally consistent hash function and a routing protocol to ensure that participating peers can efficiently route queries to peers maintaining the requested files.

1.1 Motivation

P2P systems are inherently heterogeneous. P2P members may have different service capacities, different Internet connectivity speeds, and spend different times in the system. Heterogeneity impacts the scalability of P2P systems, their ability to serve user requests and the perceived user performance. As a result, DHT-based P2P systems need to be engineered to accommodate heterogeneity in order to avoid (1) congestion at small-capacity peers, and (2) constant node join and departure that disrupt the normal system operation. This then brings out the following two issues.

1.1.1 Routing in Relaxed DHT Overlay Structure

While the structuredness of DHTs enables P2P systems to search in a diameter-bounded, content addressable overlay network, the consequence is that peer nodes are required to select and maintain a nearly fixed number of overlay neighbors deterministically based on their hash value obtained from a consistent hash function, regardless of the heterogeneity among nodes' service capacity, Internet connectivity, lifetime in the system, or even willingness to serve. As a result, more capable and/or more reliable peers cannot take more responsibility in routing and connecting of the overlay network because of the design philosophy inherent in DHTs. This limits systems' ability to exploit the full potential of its peers, and the resultant overlay network is generally susceptible to high performance degradation in the presence of high node dynamics such as frequent node join, departure and failures. DHT literature also refer node dynamics in a DHT system as *churn*, and we will use these two terms interchangeably through this dissertation.

Another option is to relax the DHT overlay structure, by allowing nodes a degree of freedom in establishing as many connections with other nodes based on their capacity and times spent in the system. Nodes then can select additional neighbors by contacting a bootstrap server, by eavesdropping to query messages, or by picking arbitrary locations in the system's coordinate space and connecting to the node managing these locations. While offering nodes more flexibility in picking their neighbors enables a more efficient and fault-resilient overlay network, doing so renders the structuredness of DHT systems and causes irregular DHT overlays. As a result, Greedy routing that leads to shortest overlay routes in regular DHT overlays, fails short of identifying shortest routes in irregular overlays. This problem motivates us to research an efficient, light-weight routing protocol for routing in irregular DHT overlay.

1.1.2 Contradiction in between C/S and P2P Architecture

Measurement studies [50, 22] indicate a high rate of node dynamics in deployed P2P systems such that most peers frequently depart and then rejoin the system while only a few remain in the system for a long time. In order to ensure an adequate level of service availability and performance under churn, a DHT node needs to monitor and maintain the connectivity of its overlay and the availability of its stored content. This maintenance

requires each DHT node to constantly probe other nodes, to replicate/code/migrate stored content, and is expensive in the presence of high node dynamics. While P2P users offer free but unstable service capacity, with stable nodes being relatively scarce, a DHT system can either rely on a small set of stable nodes with limited collective capacity, or exploit a larger set of potentially unstable user nodes and suffer maintenance and data redundancy overhead. By far to improve DHT functionality in the presence of high node dynamics, researchers have taken different approaches, whether by (1) improving node stability through a selection of the most stable nodes as DHT members, or by (2) improving the availability of the maintained data objects through replication and erasure coding. We are not aware of any study that offers an easy approach to assess the tradeoff between the stability and scalability in DHT-based P2P systems, in the presence of the system's bandwidth and storage constraints, the skew in the size and popularity of maintained content, and the workload imposed on the system. Without this knowledge, we cannot be sure if a DHT system is utilized or how to adjust the system toward a better performance.

1.2 Contributions

In this dissertation we make the following contributions:

- We show that in randomized DHTs, routing decisions based on information about direct overlay neighbors do not necessarily lead to shortest overlay routes, and that the performance gap increases in tandem with the churn rate. On the other hand, collecting routing tables of more nodes is not feasible due to the high propagation, storage and forwarding costs. We propose a novel routing protocol, RASTER, which to the best of our knowledge, is the first overlay routing protocol that encodes and aggregates routing information. Its simple bitmap-encoding technique together with the RASTER routing algorithm allow nodes to improve their forwarding overhead to a constant number of bitwise operations. The ability to process and aggregate bitmaps allow nodes to efficiently maintain shortest-path routes to remote areas, and leads to an an average path length close to optimal and better resilience to churn than current routing protocols. Also, nodes can tune RASTER parameters depending on their capacities and/or their willingness to contribute to the system.

- We make the case that careful design of DHT-based P2P systems can optimize their utility even in the presence of unstable peer nodes. While the past DHT literature studies node selection, content selection, and data redundancy strategy independently, we show that the combination of them is the major factor in DHT administration. Furthermore, our study is the first that points out the importance of accounting for the system's response time in the measure of the DHTs' data availability. While have shown the evolution trends of the system parameters leading to the optimal DHT throughput, we also provide a system-level analytical model that can be easily adapted by system administrators and researchers to access the tradeoff between the stability and the scalability in DHT-based P2P systems.

- P2P systems generally have been perceived as scalable with the system load. We show that the P2P's scalability is mostly determined by the system's response time. We highlight the performance limitations of P2P systems as expressed in terms of the system throughput and response time to user requests in the presence of node instability. Our results challenge common wisdom about P2P systems, their appropriateness for different applications, and their virtually unlimited capacity.

1.3 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 describes the background of DHTs. We present a survey on some well-known deterministic and randomized DHT systems. In Chapter 3 we present a routing protocol, RASTER protocol, that uncovers shortest overlay routes in randomized DHTs. In Chapter 4, we provide an analytical model that captures the tradeoff between the stability and the scalability in P2P systems. We use the model to demonstrate that P2P instability can be mitigated and system throughput can be optimized through careful selection of serving members and data redundancy parameters. We finally address our conclusions and future work in Chapter 5.

Chapter 2

Background of DHTs

DHTs organize nodes in a large, virtual identifier space and hash data objects to the identifier space through consistent hashing [26]. Responsibility for maintaining the mapping from data to identifiers is distributed among the nodes, in such a way that a change in the set of participant nodes causes a minimal amount of disruption in the system. This allows DHTs to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures. At any point in time, the entire identifier space is dynamically partitioned among all the participant nodes in the system such that every node owns its individual, distinct zone within the identifier space. Each DHT node then is responsible for maintaining, and answering queries about objects that are hashed into its zone. DHT-based systems typically target regular, fixed-degree overlay structures. Nodes communicate over a sparse overlay, where each node is only aware of a few other nodes, called its *neighbors*, and use them to reach other nodes in the system. Data are stored in the DHT in the form of identifier-value pairs, and any node can efficiently retrieve the desired data by querying for its associated identifier. Queries and messages from a node s for an identifier d are greedily forwarded to the neighbor that is closest to d in the system's identifier space. Greedy routing makes a locally optimal choice hoping that it will lead to shortest path routes. Shortest path routes are desirable to reduce the number of expensive overlay hops especially in systems populated by a large number of nodes.

2.1 Deterministic Distributed Hash Tables

DHTs provide data-to-identifier mapping through hash-table-like primitives. The core operation supported by DHTs is *lookup(identifier)*, which returns the identity of the node hosting the identifier; identifier-based primitives such as *get(identifier)*, *put(identifier, value)* or *remove(identifier)* [17] are built upon this core lookup operation, allowing nodes to locate, insert or remove objects. DHTs use consistent hashing [26] to uniformly map nodes and objects to an identifier space. This is done by assigning each node and each object an m-bit identifier, termed the *node identifier* (node ID) and the *data identifier* (object ID) respectively, through the use of a hash function such as SHA-1 [39]. The entire DHT identifier space is dynamically partitioned among nodes such that each node owns a distinct set of identifiers termed its *zone*, and the data objects mapped to its zone. Traditional *deterministic* DHTs [43, 51, 56, 49, 32] organize nodes into a regularly structured overlay where each node maintains a routing table containing pointers to its overlay neighbors. A node locates an data object by forwarding a lookup query to one of its overlay neighbors whose overlay position is closest to the object ID. This process repeats either recursively or iteratively, depending on the implementation, until the query reaches the node hosting the object's identifier. Upon receiving the lookup query, the node hosting the object's identifier replies the query source with the object's location, i.e. the IP addresses of nodes that possess a copy of the desired object.

DHTs construct their overlay networks largely by emulating geometry of existing parallel interconnect network such as torus, de Bruijn graph, and etc. To a great extent, the underlying geometry determines the diameter-degree tradeoff of the DHT overlay and hence its routing performance. The diameter-degree tradeoff refers to the asymptotic tradeoff between the network diameter and the average node degree - the former refers to the maximum of the minimum overlay hops between any two DHT nodes whereas the later refers to the average number of overlay neighbors maintained per DHT node. The diameter poses an upper bound on the DHT's *lookup path length* or the number of overlay hops used per query, which is an important metric that measures a DHT's routing performance. The node degree indicates the overhead expense for maintaining the DHT overlay structure. In this chapter we survey some DHT systems from the perspective of routing geometry.

One thing to notice is that even though DHTs may be built on different routing

geometry, they do adapt similar designs to maintain the overlay structure as nodes join and leave the network. First, as a node joins the DHT system, the join node obtains IP addresses of a few existing DHT nodes from a well-known bootstrap server. The join node then chooses one of them as its *gateway node* to send out a join message toward the join node's node ID. Eventually the join node gets in touch with the joint node, which is the DHT node that currently maintains the join node's node ID. These two nodes then manage to split the joint node's identifier zone. The join node then obtains a copy of the joint node's neighbor list and establishes proper overlay connections as required by the DHT geometry. Second, DHT nodes detect their overlay neighbors' presence by communicating periodic probes with their neighbors. This probing strategy is accompanied with a certain TIMEOUT mechanism such that if a DHT node does not receive any probing reply from an overlay neighbor within the TIMEOUT period [57], the DHT node will initiate a repair algorithm to find a new DHT neighbor. The TIMEOUT event also triggers a repair process which will select a neighbor of the failed DHT node to take over the failed node's zone in the identifier space.

2.1.1 Plaxton Mesh

A *Plaxton mesh* [42] connects nodes in embedded spanning trees, each associated with a data object. Each spanning tree points toward an unique node called *root node*, which is associated with the object's ID and responsible for maintaining all its associated objects. The root node of a spanning tree is selected by a globally consistent algorithm. Suppose nodes and objects are hashed to an identifier space say of length M and base b. Each Plaxton node maintains a routing table of $\log_b M$ levels, each level of b entries. The node with node ID X maintains its routing table so that the jth entry of the ith-level routing table stores the node ID and IP address of the closest node among those whose identifier has the ith digit equal to j and matches X's $(i-1)$-digit suffix. The closeness of two node is typically measured in terms of their end-to-end delay, and the smaller their end-to-end delay is, the closer the nodes are to each other. Plaxton nodes use their routing table to incrementally route a query to the root node of its destination ID. Assume a query destined for ID W arrive at node X. Node X answers the query if X is the root node of W. Otherwise, given that X and W share a h-digit suffix, node X forwards the message to its neighbor recorded in the W_{h+1}th entry of its $(h+1)$th-level routing table, where W_{h+1}

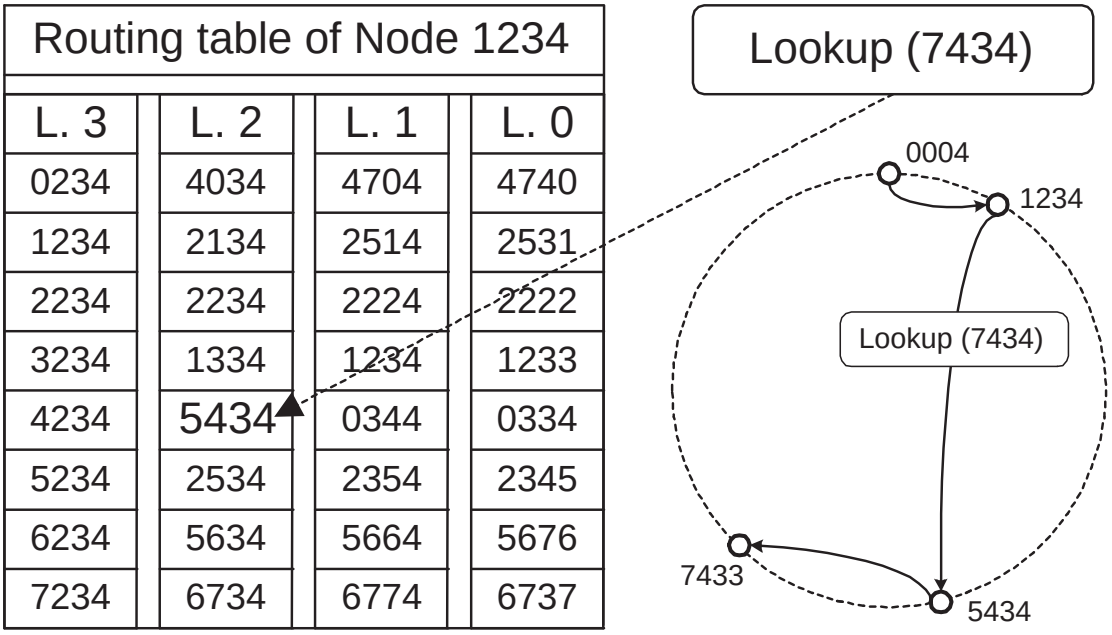

Figure 2.1: Routing in a Plaxton mesh of size 8^4.

is the value of the $(h+1)$th digit of ID W. A Plaxton mesh of n nodes and using base b can guarantee a lookup performance of $O(\log_b n)$ overlay hops by maintaining a per-node routing table of $\Theta(b \log_b n)$ entries. Figure 2.1 gives an example of routing a lookup query for ID 7434 in a Plaxton mesh of length 4 and base 8. Assume the query arrive at node 1234. Since the length of the longest shared suffix between the current hop's ID 1234 and the destination ID 7434 is 2 and the value of the 2nd digit of the destination ID is 4, node 1234 forwards the query to the node recorded in the 4th entry of its 2nd-level routing table, which is node 5434. Node 1234's routing table is shown on the left side of the figure. The complete route of the lookup query from source node 0004 to the destination's root node 7433 is shown on the right side.

Tapestry and Pastry [56, 49] are two deterministic DHTs built upon the Plaxton mesh structure, with additional mechanisms to augment scalability, fault resilience and locality adaptation. The original Plaxton mesh uses a global algorithm to achieve a consistent mapping between objects and their root nodes, which is the bottleneck to the Plaxton mesh's scalability. To resolve this problem, Tapestry instead determines an object's root node as the last hop of *surrogate routing*. Surrogate routing is almost identical to Plaxton

routing algorithm except that when the desired routing entry is absent, surrogate routing makes a deterministic forwarding decision (for example, forwarding to the neighbor whose ID is numerically closest to the destination ID). Pastry, on the other hand, requires each node to maintain an additional *leaf-set* routing table containing a set of nodes numerically closest to itself. By doing so, with high probability all Pastry nodes that are numerically closest to an object ID would identify the same node from their leaf table set as the object's root node. Each Tapestry or Pastry node also maintains a *neighborhood-set* routing table containing a set of nodes in proximity measured by their end-to-end delay, and uses them to optimize the regular routing table. Both Tapestry and Pastry nodes construct their routing tables when they join the network. On the join path to its node ID, the join node can copy and optimize each level of its routing table at each intermediate hop. This is because according to Plaxton routing algorithm, the ith hop on the route to the join node's ID must share with the join node's ID a suffix of at least length i; hence, the join node and the ith hop must share at least i levels of their routing tables. In addition, the join nodes can copy the neighborhood-set table from the gateway node because the gateway node is typically in proximity to the join node. To repair a failed entry, both Tapestry and Pastry nodes search their neighbors' routing tables for a substitute entry.

2.1.2 Torus

The Content Addressable Network (CAN) [43] resembles a d-dimensional torus by hashing nodes and objects to a d-dimensional Cartesian coordinate space. The entire space is dynamically partitioned among nodes such that each node owns its distinct coordinate zone. Each node maintains objects with hash coordinates mapped to its zone and a routing table that holds the IP address and coordinate zone of each of its overlay neighbors. In a d-dimensional coordinate space, two nodes are neighbors if their zones overlap along $d-1$ dimensions and abut along one dimension. To route a message to its destination coordinates, a CAN node uses its routing table and greedily forwards the message to the neighbor with the coordinates closest to the destination. In theory, a d-dimensional CAN achieves a lookup performance of $O(d \cdot n^{\frac{1}{d}})$ hops by maintaining $\Theta(d)$ neighbors; in a special case that $d = \log n$, the CAN has a lookup performance of $O(\log n)$ hops. Because its routing-table size is independent of the network size, a CAN system requires additional efforts to match its routing-table size to the network size to achieve a satisfactory lookup performance.

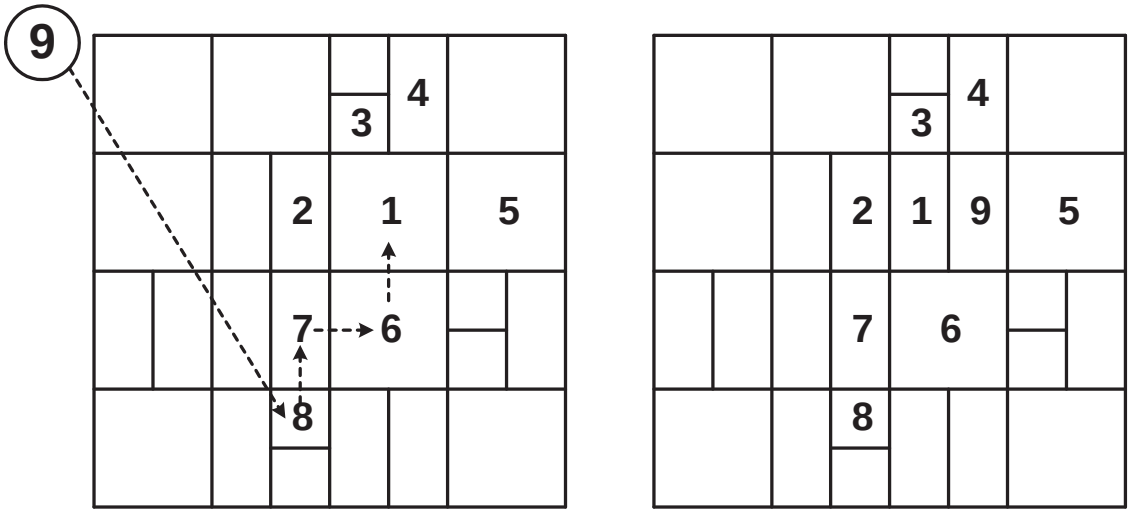

Figure 2.2: A 2-dimensional CAN network.

Figure 2.2 shows an example of how node 9 joins a 2-dimensional CAN. In the left figure, node 9 routes a join message through the gateway node, node 8, to its coordinates that are contained within node 1's coordinate zone. Eventually Node 9 reaches node 1, then they manage to split node 1's zone and update their overlay connections accordingly. The resultant CAN topology after node 9's join is illustrated in the figure to the right.

A torus is simply a mesh with wrapped-around edges; its formal definition is given as follows.

Definition 1 *A d-dimensional torus of $\prod_{i=1}^{d} k_i$ nodes, which has k_i nodes along dimension i, is a directed graph with node set $V = \{a = (a_1 \ldots a_d) | a_i \in [0, k_i - 1] \ \forall \ 1 \leq i \leq d\}$ and edge set $E = \{(u, v) | u, v \in V\}$ iff there exists exactly one j such that $u_j = (v_j \pm 1) \bmod k_j$ and that $u_i = v_i \ \forall \ i \neq j$, $1 \leq i, j \leq d$.*

The CAN system relies on the following mechanisms to handle node dynamics. A new node that joins a CAN must claim its zone in the coordinate space. Having obtained its

coordinates by consistent hashing, the join node routes a join message through a gateway node to reach the owner of its coordinates. The join node then claims an half of the owner's zone and obtains a copy of the owner's routing table. The join node and owner node then update their routing tables accordingly based on the overlay topology after the zone split. When detecting the departure of its neighbor, a CAN node initiates a takeover timer whose value is set proportional to the node's zone size. When the timer expires, the node sends a takeover message conveying its zone size to all neighbors of the failed node. The alive neighbor with the smallest zone size in turn takes over the failed node's zone in the coordinate space. Greedy forwarding may temporarily fail nearby the failed node before the failed node's zone can be successfully taken over. In this case, a CAN node either postpones any forwarding that goes through the failed zone or uses the expanding-ring technique to search for a substitute next hop.

2.1.3 Chord Ring

Literature perceives Chord's overlay geometry [51] as a ring because it orders identifiers on a circular, ring-like identifier space. However, the overlay connections of each Chord node indeed space in a geometric-sequence distance across the identifier ring. Chord hashes nodes and objects to the identifier ring, and associates each object with its ID's *successor*. An object's successor is the first node encountered clockwise from its ID along the identifier ring. In a base-b Chord on a m-bit identifier space, a node with ID X maintains a routing table termed its *finger table* so that the ith finger points to the successor of identifier $X + (\frac{b}{b-1})^{i-1}$, $1 \leq i \leq \log_{\frac{b}{b-1}} 2^m$. A node's first finger is also called its *successor finger* because it always points to the successor of its node ID. A Chord node locates an object as using its finger table to find out the successor of the object's ID. When a lookup query for ID W arrives, node X answers the query with its successor's location if W falls between X and X's successor's node ID. Otherwise, X forwards the query to the node pointed by its finger whose node ID mostly immediately precedes W. Assume W fall in between the node ID recorded in the kth and $(k+1)$th finger of node X. Because the distance between W and X is at most $(\frac{b}{b-1})^k$ and the distance between W and the next hop, which is pointed by node X's kth finger, is at most $(\frac{b}{b-1})^k - (\frac{b}{b-1})^{k-1} = (\frac{1}{b})(\frac{b}{b-1})^k$. Therefore, each single hop on a Chord's routing path reduces the lookup distance by a factor of at least $\frac{1}{b}$. This means, a base-b Chord achieves a $O(\log_b n)$-hop lookup performance by

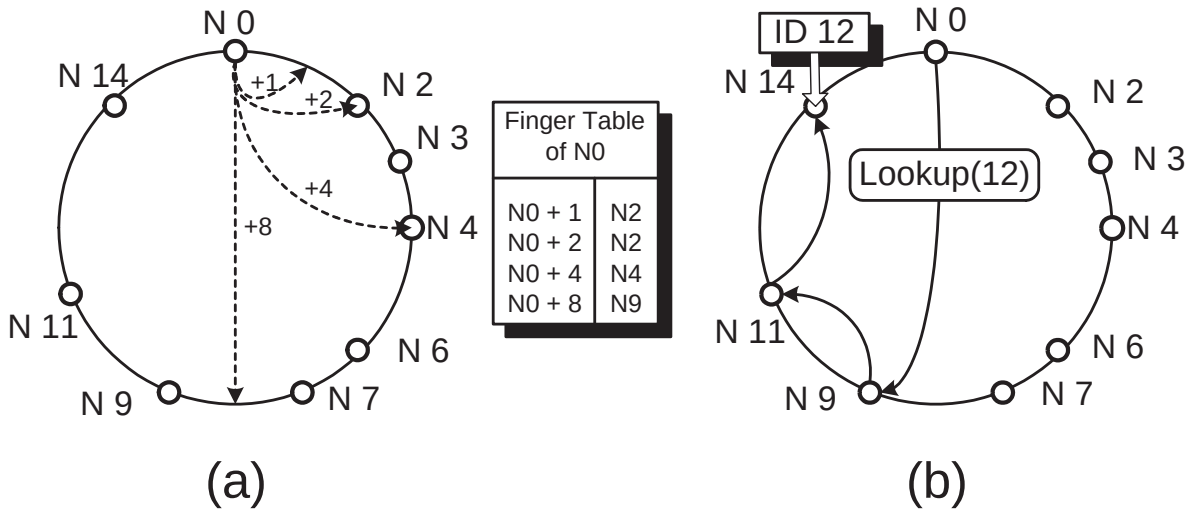

Figure 2.3: A Chord system that uses base 2 and a 4-bit identifier space.

maintaining $\Theta(b \log_b n)$ fingers per node. Figure 2.3 gives an example of a Chord that uses a 4-bit, base-2 identifier space. Figure 2.3-(a) gives an example of Node 0's finger table in which the ith finger points to the successor of identifier 2^{i-1}, $1 \leq i \leq 4$. Figure 2.3-(b) gives a sample route from node 0 toward ID 12.

Routing by only nodes' successor finger is sufficient to guarantee a correct lookup for any identifier. If Chord nodes keep their successor finger up to date, each of them can incrementally correct their faulty fingers by periodically querying for the successor of each finger's associated identifier. Now we explain how nodes keep their successor finger up to date as nodes join and leave the network. When a new node X joins a Chord network, it acquires the successor of its identifier, say node S, through a gateway node. Node X then notifies node S the predecessor-and-successor relationship between them. As a result of this process, node X could become the new successor of certain objects maintained by node S. If this occurs, node S needs to transfer the corresponding objects and their information to node X. At this moment, the old predecessor of node S, say node P, should point its successor finger to node X instead of node S. Figure 2.4-(a) illustrates the inconsistency that occurs during this join process: by Chord's join process, node X and node S correctly set each other as the predecessor and successor, but node P's successor finger still points to node S, which should be corrected by pointing it to node X. To resolve this ring-inconsistency incidence, each Chord needs to periodically executes a process called *stabilization*, by which

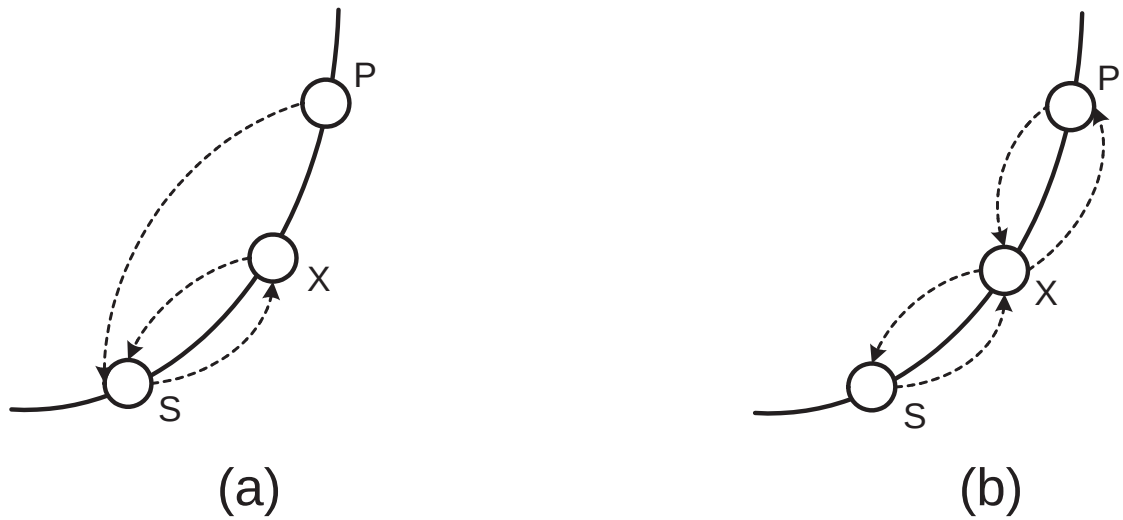

Figure 2.4: Inconsistency in Chord's finger tables due to node join and departure.

a node learns from its successor the up-to-date predecessor of its successor and then uses this information to update its successor finger. Figure 2.4-(b) illustrates the stabilization process: if each Chord node periodically hears from its successor the identity of the predecessor of its successor, when the inconsistency stated in (c) occurs, node P sooner or later will know from node S that node S's predecessor is indeed node X. Then, node P will contact node X, and both of them will realize the predecessor-and-successor relationship between them. Another possible ring-inconsistency incidence could occur when multiple nodes fail simultaneously, which makes certain nodes lose track of their successor. This can be resolved by having each Chord node maintain an additional *successor list* that contains its first $\Theta(\log n)$ successors, and the failed successor finger can be substituted by the most immediate and alive node in the successor list. It has been shown in [40] that for each period the network doubles or halves in size, lookups perform well in Chord as long as each node uniformly executes $\Omega(\log^2 n)$ rounds of stabilization.

2.1.4 de Bruijn Graph on Ring

D2B, Koorde, ODRI and Distance Halving [18, 25, 32, 38] are DHTs that hash nodes and objects to an identifier ring and organize nodes into a de Bruijn graph. A de Bruijn graph of degree d and diameter k, $\overrightarrow{B}_d^k$, is a directed graph with node set $V = \{a = (a_1 \ldots a_k) | a_i \in [0, d-1] \ \forall \ 1 \leq i \leq k\}$ and edge set $E = \{(u, v) | u, v \in V, \ v =$

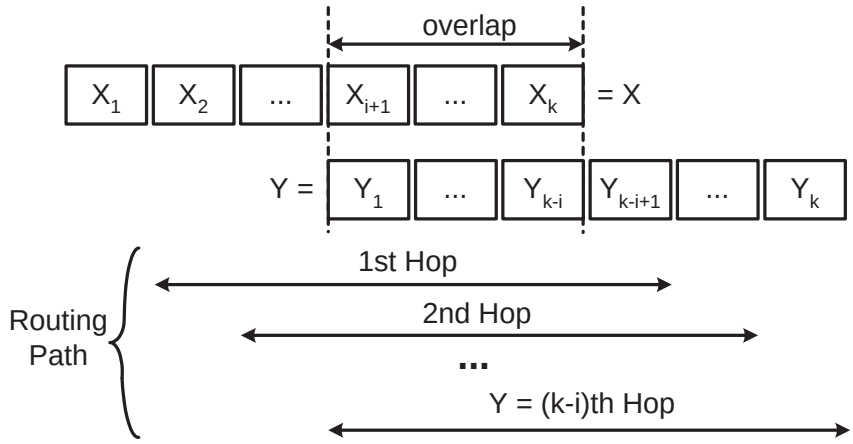

Figure 2.5: The optimal route from Node X to Node Y in a de Bruijn graph of diameter k, assuming X's suffix and Y's prefix share an i-digit sequence.

$dx + j \ \forall \ j \in [0, d-1]\}$. Given that node X's suffix and node Y's prefix share an i-digit sequence, the optimal route from node X to node Y uses exact $(k-i)$ hops and follows a specific *de Bruijn sequence*. In this case, the de Bruijn sequence is the k-digit shuffle sequence of $\{X_2X_3 \ldots X_{k-1}X_kY_{i+1}Y_{i+2} \ldots Y_{k-1}Y_k\}$. The optimal de Bruijn route are illustrated in Figure 2.5. One way of emulating a de Bruijn graph in a DHT is by treating each identifier as a virtual de Bruijn node and associating each virtual de Bruijn node with its successor. DHTs of size n that emulate a de Bruijn graph of degree d achieve a $O(\log_d n)$-hop lookup performance by maintaining a per-node routing table of $\Theta(d)$ entries. The remained details of designing a de-Bruijn-based DHT are similar to those used in Chord such as the mechanisms used to used to keep fingers consistent and up to date.

2.1.5 Butterfly Graph on Ring

Viceroy [33] is a deterministic DHT that emulates a butterfly graph on an identifier ring. Viceroy hashes objects and nodes to an identifier ring, and associates each object with its identifier's successor. Each Viceroy node is assigned one of $\log n$ levels at random when it joins the network. The network consists of three sets of overlay links: those of a general ring, of local level rings, and of the butterfly structure. A general ring connects each node with its predecessor and successor on the identifier ring; each level ring connects nodes with their predecessor and successor on the same level; the butterfly connects node X on

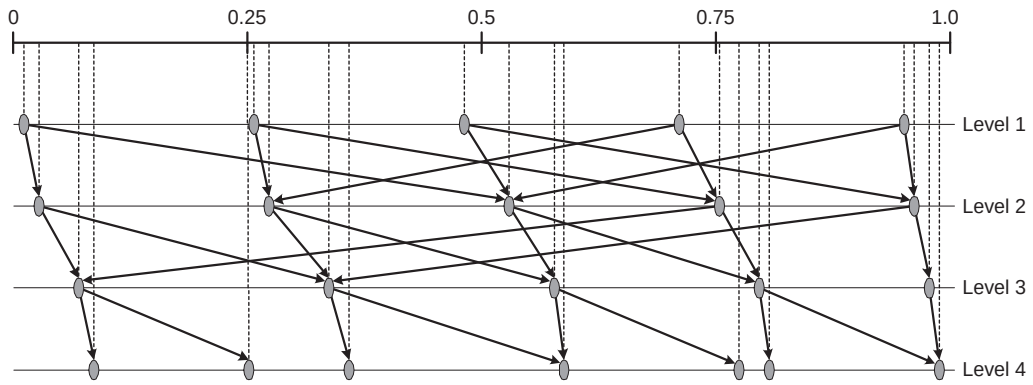

Figure 2.6: A Viceroy network. For simplicity, lines from the top to the bottom assemble the general ring and level-1 to level-4 rings. Up-level links are not shown in the figure.

level l to the nodes on level $l+1$ that are clockwise closest to identifier X and to identifier $X+2^{-l}$ if $l < \log n$, and to the node on level $l-1$ that is clockwise closest to identifier X if $l > 1$. Routing in a Viceroy network consists of three phases: the first phase uses $O(\log n)$ hops to move toward the first level; the second uses $O(\log n)$ hops to traverse down the levels until the down-link is null or overshoots the destination identifier; the third uses the successor/predecessor neighbors of the general ring or level rings to reach the destination in another $O(\log n)$ hops. A sample Viceroy network is shown in Figure 2.6. Other design details of Viceroy are similar to those used in Chord. Viceroy achieves a $O(\log n)$-hop lookup performance by maintaining 7 overlay connections at each node. However, because of the hidden constant in Viceroy's asymptotic performance bound, its actual lookup path length is much worse than those of other DHT geometry.

2.1.6 Problems of Deterministic DHTs

Deterministic DHTs organize their nodes in a hash space by emulating a family of interconnection networks characterized by an efficient diameter-degree tradeoff. Table 2.1 summarizes the diameter-degree tradeoff of previously introduced DHT geometry. However, measurement studies have shown that applications targeted by DHTs have a very dynamic membership and there exists significant heterogeneity in their participant nodes bandwidth capacity [50, 22, 9, 29]. Furthermore, today DSL or cable modem service typically uses a scheme called Network Address Translation (NAT) to allow an office/household group to share a single IP address. While most DHT nodes access the Internet through DSL or

Table 2.1: Asymptotic diameter-degree tradeoff of various DHT geometry.

DHT Overlay Geometry	Degree	Diameter
Plaxton mesh	$b \log_b n$	$\log_b n$
Torus	$2d$	$\frac{1}{2} d n^{\frac{1}{d}}$
Chord Ring	$(b-1) \log_b n$	$\log_b n$
de Bruijn	d	$\log_d n$
Viceroy Butterfly	7	$2 \log_7 n$

cable modem service, nodes using NAT can connect to those outside their local network but nodes outside the NAT hosts' local network cannot connect to them. Hence, regular overlay structures prevent DHTs from utilizing the collaborative capacity of the whole network and are problematic to implement in the real world.

2.2 Randomized DHTs

Randomized DHTs [34, 11, 55, 24, 23] aim for remedying the problems of deterministic DHTs by offering nodes certain flexibility in selecting their neighbors. Literacy shows that this flexibility in neighbor selection leads to better overall performance in terms of path latency, static resilience and local convergence [23, 13]. Most randomized DHTs are inspired by Kleinberg's small-world model [28]. Therefore, this section begins with a brief introduction to Kleinberg's small-world model and then summarizes several randomized DHTs built on the small-world model. We then discuss routing algorithms used in randomized DHTs and their routing performance.

2.2.1 Small-World Model and Randomized DHTs

The small-world phenomenon was first inaugurated in social-science experiments conducted by Stanley Milgram and his co-workers in the 1960's [37, 52]. This phenomenon demonstrates that connections to random neighbors lead to a network where two individuals can reach each other through short chains of acquaintances. In 1998, Watts and Strogatz developed a model for the small-sorld phenomenon based on a class of random networks in which the network's edges are divided into local and long-range contacts [53]. Kleinberg later proposed a framework that encapsulates the paradigm of Watts and Strogatz in a two-dimensional grid network [28], known as Kleinberg's *small-world model*. In Kleinberg's model, nodes in the network are labeled with the set of $n \times n$ grid points, $\{(i,j) : i,j \in \{1,2,\ldots,n\}\}$, and the the distance between two nodes $u = (i,j)$ and $v = (k,l)$ is defined as $d(u,v) = |k-i| + |l-j|$. Each node u has directed local connections to nodes within constant distance p, and is equipped with q directed long-range connections, each pointing to an arbitrary node v with probability proportional to $d(u,v)^{-r}$, $r \in Z^{-*}$. Kleinberg proved that using only local information for routing, $r = 2$ is the only value for which there exists a decentralized routing algorithm capable of achieving the optimal performance.

Symphony, Skip-nets, randomized Hypercube, randomized Chord are randomized DHTs inspired by Kleinberg's small-world model. Symphony [34] hashes nodes and objects to an unit identifier ring, which is the unit identifier interval $[0,1)$ that wraps around, by a mechanism similar to Chord's [51]. Each Symphony node has local connections to its

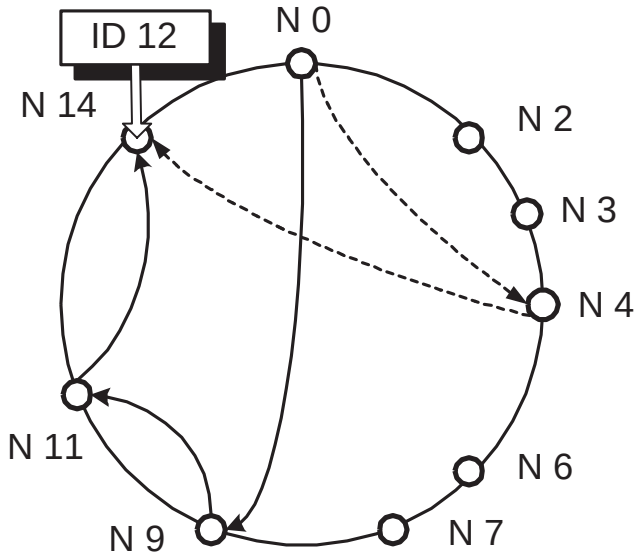

Figure 2.7: Greedy forwarding not necessarily lead to the shortest-path route to the destination.

predecessor and successor on the ring, and maintains $k \geq 1$ long-range connections. Each long-range connection points to a node hosting the identifier that is x points away, with a probability that follows a probability density function harmonically distributed over x. Skip-nets was proposed by Harvey et al in [24], which organizes nodes on a basic ring structure as well. In a skip net each node chooses a random binary string as its node identifier, and determines where its long-range connections should point to by using the randomness in its node identifier. Randomized Hypercube and randomized Chord are built upon the framework of CAN and Chord [43, 51], except that each node is equipped with a few long-range connections that connect two nodes with probability inversely proportional to their overlay distance. The diameter of small-world randomized DHTs on n nodes, each of out-degree $\Theta(\log n)$, is known to be $O(\frac{\log n}{\log \log n})$ with high probability [14, 35].

2.2.2 Greedy Routing in Randomized DHTs

Randomized DHTs obtain better overall performance by relaxing the restriction on nodes' neighbor selection. Nevertheless, such relaxation results in an irregular overlay where Greedy routing, the prevalent routing strategy in deterministic DHTs, fails short of identifying the shortest chains of hops between nodes. Greedy routing algorithm is formally

defined as follows:

Definition 2 *Assume the message is routed to destination Y and is currently at node X, $X \neq Y$. Let $W_1, W_2, \ldots, W_k$ be the neighbors of X. By Greedy routing, X forwards the message to one of its neighbor W^* such that the distance from W^* to Y is the smallest among those of W_i, $1 \leq i \leq k$.*

By Greedy routing, a node perceives its neighbor closest to the destination as the one leading to a shortest chain of hops between them. This is true in the case of deterministic DHTs because they intend to place nodes equally spaced in the overlay. Therefore, the hop-count length of a Greedy route between two nodes is proportional to the overlay distance between them. For randomized DHTs, due to randomness in nodes' neighbor selection, the neighbor leading to the shortest-path route is not necessarily the one selected by Greedy routing algorithm. For example in Figure 2.7, consider a randomized Chord using base 2 and a 4-bit identifier space, where each node is equipped with one long-range connection that can point to other nodes at random. Assume the message destined for ID 14 currently arrive at node 0. While the Greedy next hop, node 9, leads to a route of 2 overlay hops by following the regular Chord connections, node 4 is actually a better next hop choice since it leads to a route of 1 hop because its long-range connection servers as a shortcut to the node that hosts ID 14. The length of Greedy routes between arbitrary nodes in a small-world randomized DHT of degree $\Theta(\log n)$ is $\Theta(\log^2 n)$ hops with high probability [8, 28], even though the network diameter is only $\Theta(\frac{\log n}{\log \log n})$ hops.

2.2.3 Neighbor-of-Neighbor Greedy Routing in Randomized DHTs

Manku and et al. proposed Neighbor-of-Neighbor(NoN)-Greedy routing for discovering the shortest-path route in randomized DHTs [35]. By NoN routing, nodes maintain not only information about their overlay neighbors but also about the neighbors of their neighbors, and its routing algorithm is formally defined as follows:

Definition 3 *Assume the message is routed to destination Y and is currently at node X,*

$X \neq Y$. Let $W_1, W_2, \ldots, W_k$ be the neighbors of X. For each W_i, $1 \leq i \leq k$, find Z_i - the closest to Y among W_i and its neighbors. Let Z^* be the closest to Y among $Z_1, Z_2, \ldots, Z_k$. By NoN routing, X forwards the message to W_i.

In a randomized DHT of n nodes and when the average node out-degree is $\Theta(\log N)$, NoN routing leads to an average number of $\Theta(\log N / \log \log N)$ overlay hops between arbitrary nodes with high probability. However, this improvement in performance comes with the following costs. First, with NoN routing, neighboring nodes need to exchange their full-size routing tables, resulting in the propagation overhead proportional to $\Theta(\log^2 N)$ routing entries. Second, the NoN routing table is of $\Theta(\log^2 N)$ entries as opposed to the Greedy routing table of $\Theta(\log N)$ entries. Hence, the complexity of NoN routing algorithm is $\Theta(\log^2 N)$, and its forwarding decision is an order of magnitude slower than that made by Greedy routing.

Chapter 3

Discovering Shortest Overlay Routes in Randomized DHTs

Randomized DHT-based Peer-to-Peer (P2P) systems bring together the collective power of its members by allowing nodes the flexibility to contribute based on their resources and/or their willingness to serve. Routing in presence of overlay irregularity, which is inherent in these systems, is challenging. Routing decisions based *only* on information about close overlay neighbors can deviate from shortest overlay routes by an order of magnitude. A more detailed information about the overlay is needed to recover this gap in performance *while* keeping the cost of exchanging, maintaining, and using this information to generate routing decisions manageable. This chapter presents our proposal of a novel routing protocol, RASTER, that uncovers shortest overlay routes between nodes in randomized DHTs.

3.1 Introduction and Background

Traditional DHT-based P2P systems [32, 43, 51, 56] target regular, fixed-degree overlay structures. In these systems, nodes select their overlay neighbors deterministically based on their relative positions in the system's hash space regardless of the heterogeneity

among nodes' service capacity, Internet connectivity, lifetime in the system, or even willingness to serve [50]. As a result, fixed-degree overlays limit P2P systems' ability to exploit the full potential of its members and are generally susceptible to high performance degradation in the presence of high churn.

To remedy these problems, *Randomized* DHT-based P2P systems [11, 23, 24, 34, 55] offer nodes some flexibility in picking their neighbors. This flexibility leads to better overall performance in terms of path latency, static resilience and local convergence [23]. Most randomized P2P systems were inspired by Kleinberg's *Small-World* model [28], which shows that connections to random neighbors lead to a network where nodes can reach each other through short chains of overlay hops. Nevertheless, Greedy routing, the prevalent routing strategy in DHT systems, fails short of identifying the shortest chains of hops between nodes. The reason being that by Greedy routing a node maintains *only* a list of its overlay neighbors and forwards queries to the neighbor that is *perceived* as being closest to the query destination. In the presence of irregularity due to randomness, the neighbor leading to the shortest chain of hops cannot be identified simply from the list of neighbors.

In [35], Manku and et al. proposed that nodes maintain not only information about their overlay neighbors but also about the Neighbors of their Neighbors (NoN). Routing in this setup is done by greedily forwarding queries to the neighbor that has a neighbor closest to the destination. It has been shown that in a randomized DHT system of n nodes and when the average node out-degree is $\Theta(\log n)$, NoN routing leads to an average number of $\Theta(\log n/\log\log n)$ overlay hops between arbitrary nodes with high probability, outperforming the traditional Greedy routing, which in general has a path length of $O(\log n)$ [11, 24, 55]. This improvement in performance comes at the cost of an order of magnitude larger routing tables ($\Theta(\log^2 n)$ entries as opposed to $\Theta(\log n)$), and thus an order of magnitude slower forwarding decision, in addition to the cost of exchanging routing tables between neighbors. Furthermore, as we make the case in this chapter, NoN does not necessarily lead the shortest overlay paths, and there is room for further improvement by maintaining even *broader* views of the overlay. However, generalizing the NoN approach by allowing nodes say to maintain the routing tables of their neighbors and the neighbors of their neighbors will certainly not scale due to the cost associated with propagating, storing and computing the forwarding decision based on all this information.

This chapter presents our proposal of a novel routing protocol, RASTER, to approximate shortest chains of overlay hops between nodes in randomized DHT systems.

RASTER is distinguished from others by its abstraction of an overlay into a collection of virtual bins (hash space partitions), each of which hosting a small number of nodes in the network (managing their corresponding hash space partition). Unlike Greedy or NoN routing that relies on full routing tables, the information exchanged and maintained to generate RASTER forwarding decision is a compact representation of routing tables as *bitmaps*.[1] This allows routing information to be aggregated, propagated more efficiently; and minimizes the cost of the forwarding decision to merely a few bitwise operations. It also provides nodes with the flexibility to adjust the resolution of the maintained bitmaps based on their storage/processing capabilities. The light-weight nature of RASTER allows nodes to more generally maintain information of the neighboring topology within a *radius* of $a \geq 1$ hops. This generalization allows RASTER to outperform NoN in approximating shortest overlay routes, with less forwarding, storage and communication overhead, and results in better resilience to system churn. The cost associated with the storage and the propagation of the bitmaps is manageable and grows logarithmically with the size of the network. RASTER achieves: $\Theta(\log n)$ forwarding complexity, which is comparable to Greedy routing but superior to NoN's $\Theta(\log^2 n)$ forwarding complexity; an average lookup path length of $\Theta(\log n / \log\log n)$ hops, comparable to NoN's but superior to Greedy routing's $O(\log n)$ hops.

The rest of this chapter is organized as follows. In Section 3.2 we define bitmaps, explain how they are constructed. We provide the details of the RASTER protocol, analyze its complexity and the tuning of its parameters. In Section 3.3 we compare the performance of RASTER to other protocols. We limit our discussion to a 2-dimensional-torus overlay similar to that of CAN [43] and allow nodes to have additional connections to arbitrary neighbors in addition to their CAN connections. This is intended to simplify the illustrations and does not imply that the proposed scheme cannot be applied on arbitrary overlay structures.

[1] Hence the name of the protocol. RASTER is a formation consisting of the set of horizontal lines composed of pixels that is used to form an image on a CRT.

3.2 Bitmaps and the RASTER Routing Protocol

3.2.1 Bitmap Definition

A bitmap is a compressed/encoded representation of a set of nodes in a DHT system. Here is a more formal definition of bitmaps.

Definition 4 *A bitmap of resolution r and covering a radius a from an arbitrary node s, $B_s^{a,r}$, is an r-bit binary string that represents the overlay positions that are reachable within a hops from s. The system's hash space is evenly partitioned into r equal-size virtual bins.*[2] *Bit i in $B_s^{a,r}$, $0 \leq i \leq r-1$, is associated with bin i, and is set to 1 if there exists at least one node in bin i that is reachable within a hops from s.*

Based on this definition, $B_s^{1,r}$ represents the overlay neighbors of node s, $B_s^{2,r}$ represents the nodes within a radius of two from s, etc. Unlike routing tables, bitmaps hide from a node the overlay details in the virtual bins, which is typically irrelevant to the node's forwarding decision. The size of a bitmap is independent of the number of nodes in the system. Furthermore, bitmaps can be easily processed and aggregated by bitwise operations such as bitwise ANDing and ORing, which significantly reduces the cost associated with making a forwarding decision.

3.2.2 Bitmap Construction

In order to obtain $B_s^{1,r}$, each node s can simply encode its set of overlay neighbors (Figure 3.1-(a)). To obtain $B_s^{2,r}$, node s exchanges $B_s^{1,r}$ bitmaps with its neighbors. As a result of this exchange, node s obtains $B_k^{1,r}$ for all neighbors k of s, which from the point of view of node s, are the bitmaps that cover neighbors that are within a radius of two. Bitwise ORing these bitmaps result in $B_s^{2,r}$ (Figure 3.1-(b)). Similarly, to obtain $B_s^{3,r}$, nodes exchange $B_s^{2,r}$ with all their neighbors, perform the bitwise ORing of $B_k^{2,r}$ for all neighbors

[2] *Bins represent partitions of the DHT's hash space. Nodes managing zones of the hash space that fall into one partition are considered members of the same bin.*

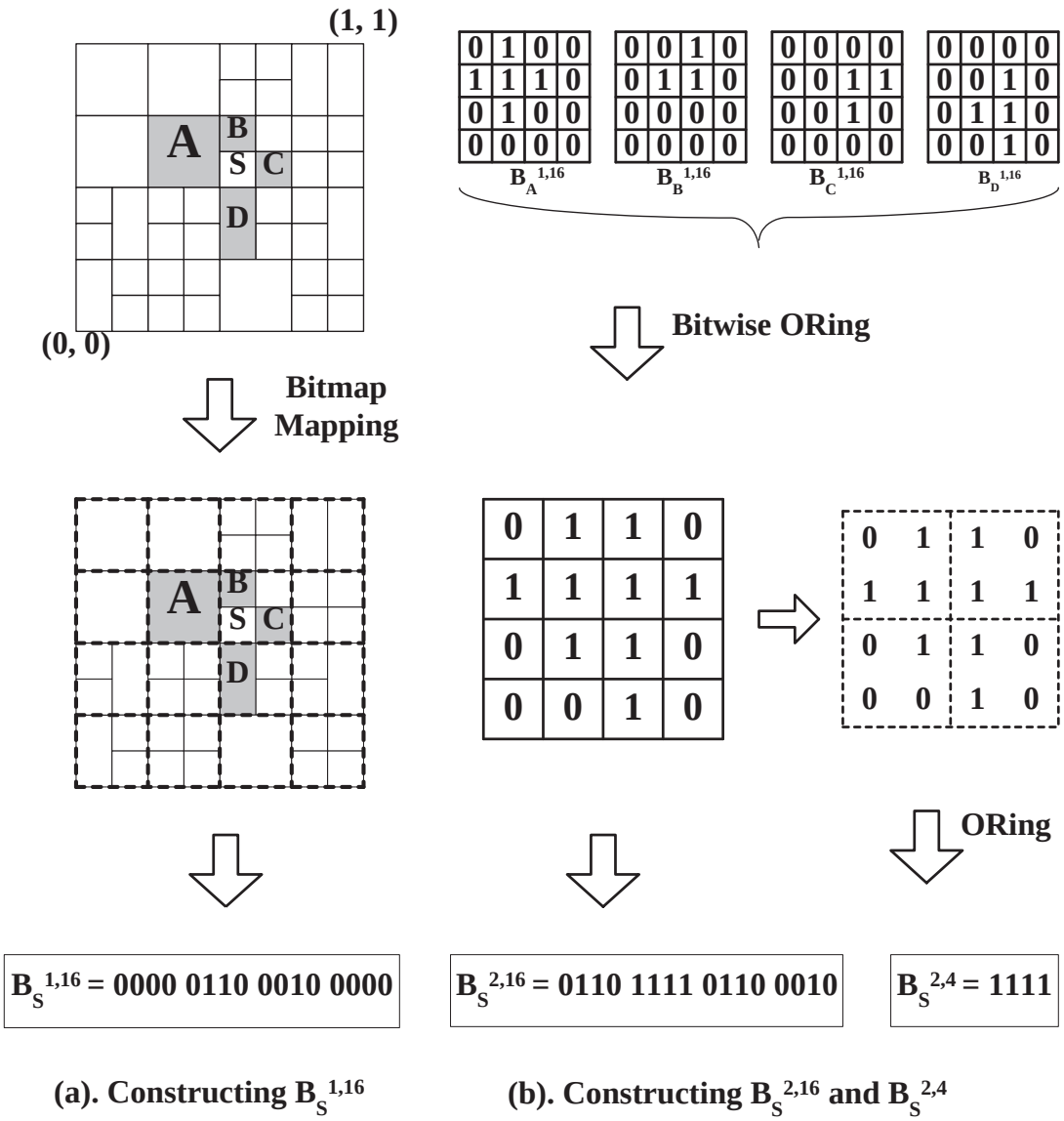

Figure 3.1: How node s generates bitmaps (a) $B_s^{1,16}$, and (b) $B_s^{2,16}$.

k to obtain $B_s^{3,r}$, etc. Note that a bitmap $B_s^{a,r1}$ can be converted to bitmaps with lower resolution, $r2 < r1$, at the cost of loosing some details (Figure 3.1-(b)). On the other hand, conversion from low to high resolution bitmaps, $r3 > r1$, can possibly induce erroneous bits in the high resolution bitmap. In general, we assume that nodes use a maximum resolution, r_{max}, which is a system parameter. The example in Figure 3.1 shows how node s generates bitmaps (a) $B_s^{1,16}$, and (b) $B_s^{2,16}$. A bitmap string from left to right is associated with the bins from left to right and from top to bottom. In (a), node s determines which bins contain its neighbors (A, B, C, D), and then sets the associated bits in $B_s^{1,16}$ to 1. In (b), ORing of all the neighbors' 1-hop bitmap leads to $B_s^{2,16}$; further binning of $B_s^{2,16}$ into 4 bins and ORing every bits within each bin leads to $B_s^{2,4}$.

3.2.3 The RASTER Routing Protocol

The RASTER protocol requires nodes to store *appropriate* bitmaps. Specifically, to implement RASTER(a), in which each node s maintains bitmaps about its neighboring topology within a radius of a, node s stores $B_s^{1,r_{max}}$, $B_s^{2,r_{max}}$, ..., $B_s^{a,r_{max}}$ in addition to $B_k^{1,r_{max}}$, ..., $B_k^{a-1,r_{max}}$ for all its neighbors k. Upon receiving a query for a key[3], node s identifies the bin in the system's hash space (and the associated bit in the stored bitmaps), b_0, corresponding to the hash value of this key. Then, node s forwards the query to its neighbor, k^*, that (1) can reach the closest bin to b_0 (2) following the minimum number of hops.

A straightforward implementation of RASTER(a) at an arbitrary node s is to search $B_s^{1,r_{max}}$, $B_s^{2,r_{max}}$, ..., $B_s^{a,r_{max}}$ to find the *closest* bit to b_0 that is set to 1 in any of the bitmaps.[4] If the closest bit is identified in $B_s^{H,r_{max}}$, then there exists a neighbor that can reach the closest bin in H hops. If more than one bitmap are identified as having the closest bit, then ties are broken by choosing the bitmap with the minimum H. Then, to find the neighbor leading to the bin closest to b_0, node s searches $B_k^{H,r_{max}}$ for all its neighbors k until one of them, k^*, has the bit corresponding to b_0 set to 1. The message is then forwarded to k^*.

However, searching the bitmaps for the closest bit to b_0 is quite expensive. This overhead can be reduced dramatically at the cost of maintaining multiple resolutions of each

[3]A key refers to a any resource depending on the system's application.

[4]The distance between bits is the distance between their corresponding bins in the system's hash space.

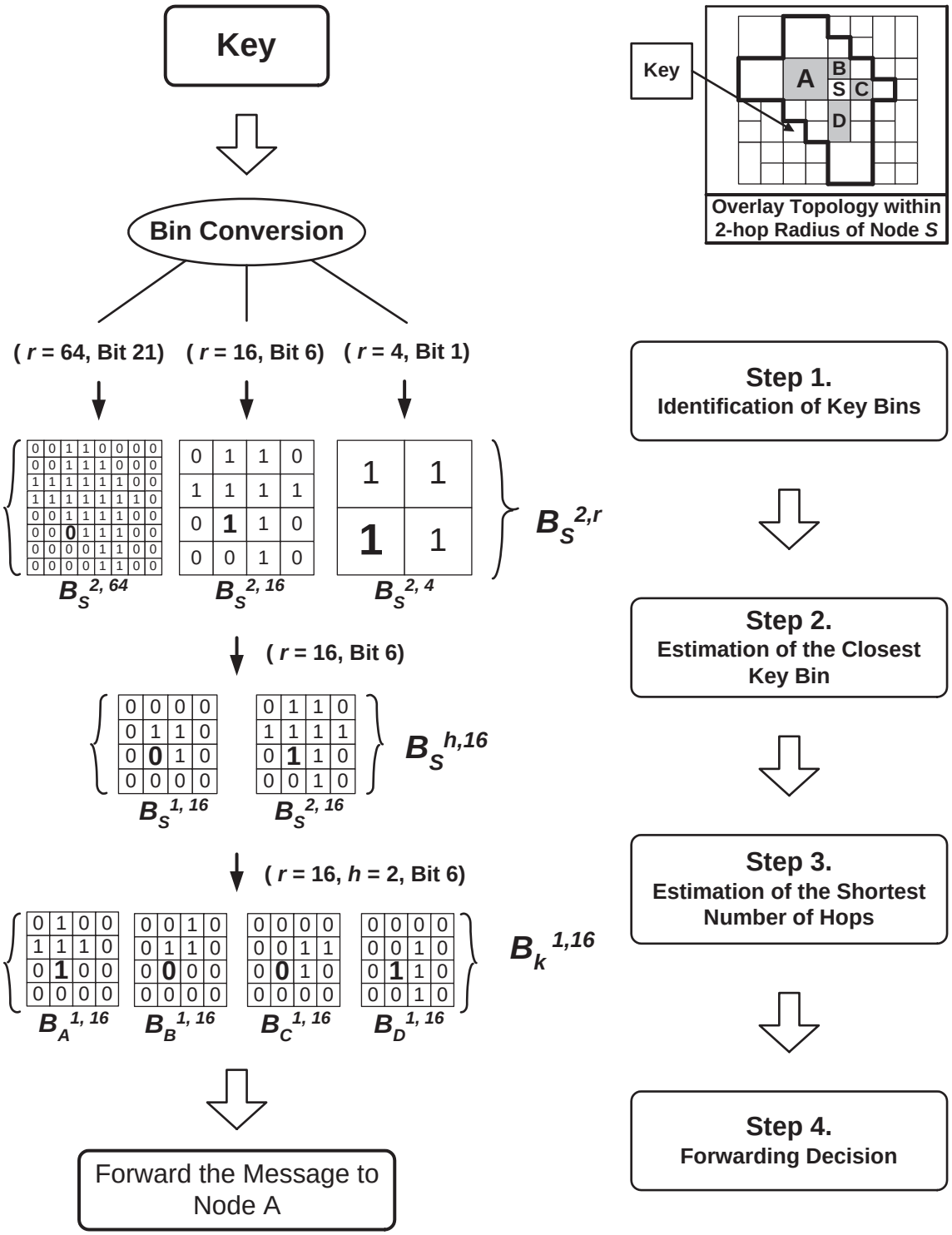

Figure 3.2: How Node s routes a query for a key following RASTER(2) protocol.

Table 3.1: Summary of complexity of NoN and RASTER routing protocols in terms of forwarding-decision, propagation and storage overheads. Assume the network size n, the maximum RASTER resolution r_{max}, and both protocols taking routing information within a hops into account.

Overhead \ Strategy	NoN (a)	RASTER (a)
Forwarding	$\Theta(\log^a n)$	$O(\log n)$
Propagation	$\Theta(\log^a n)$	$\Theta(a \cdot r_{max} \cdot \log n)$
Storage	$\Theta(\log^a n)$	$\Theta(a \cdot r_{max} \cdot \log n)$

bitmap instead of only one resolution r_{max}. Therefore, instead of searching all the bitmaps with resolution r_{max} for the bit *closest* to b_0, we search different resolution bitmaps for the highest resolution bitmap that has b_0 or the bit corresponding to b_0 set to 1. Figure 3.2 illustrates the RASTER(2) protocol in action when a query message reaches node s. The main steps in the implementation of RASTER(a) are as follows.

- **Step 1. Identification of the Key Bins.** Determines the bins where the queried key resides, for all the maintained resolutions. If this leads to a miss, then the message is forwarded by Greedy routing.

- **Step 2. Estimation of Closest Bin.** Scans $B_s^{a,r}$ in *decreasing* order of resolution r for a hit ('1') in the bit corresponding to the key's bin. Let r_{hit} be the highest resolution that leads to a hit and Bin_{close} be the bit in this resolution corresponding to the key's bin.

- **Step 3. Estimation of Shortest Number of Hops.** Scans $B_s^{j,r_{hit}}$ in *increasing* order of the radius j, looking for a hit in the Bin_{close} bit. The first radius leading to a hit, H, represents the shortest number of hops leading to the closest bin to the destination.

- **Step 4. Forwarding Decision.** Determines the next hop neighbor, k^*, whose $B_{k^*}^{H,r_{hit}}$ has a hit in the Bin_{close} bit.

3.2.4 Protocol Complexity

Consider a randomized DHT of size n and out-degree $k = O(\log n)$ implementing RASTER(a). Let l denote the total number of maintained resolutions for each bitmap stored at nodes. The *forwarding decision* overhead of RASTER(a) in the worst case is $l +$

Table 3.2: Overheads of a-hop NoN/RASTER routing protocols as the average node degree k varies, assuming the use of a 160-bit hash space and RASTER resolutions $\{16^2, 32^2, 64^2\}$. The table exhibits its forwarding-decision overhead (F.) in number of bitwise operations, propagation (P.) and storage overheads (S.) in bytes.

	NoN		RASTER (2)			NoN (3)		RASTER (3)		
k	F.	P./S. (B)	F.	P. (B)	S. (B)	F.	P./S. (B)	F.	P. (B)	S. (B)
5	25	$0.5K$	10	$2.56K$	$3.36K$	125	$2.5K$	11	$5.12K$	$6.72K$
10	100	$2K$	15	$5.12K$	$6.72K$	$1K$	$20K$	16	$10.24K$	$13.44K$
20	400	$8K$	25	$10.24K$	$13.44K$	$8K$	$160K$	26	$20.48K$	$26.88K$
50	$2.5K$	$50K$	55	$25.60K$	$33.60K$	$125K$	$2.5M$	56	$51.20K$	$67.20K$
100	$10.0K$	$200K$	105	$51.20K$	$67.20K$	$1M$	$20M$	106	$102.40K$	$130.44K$

$a+O(\log n)$ bitwise operations. The $O(\log n)$ operations are used to scan through neighbors' bitmaps or to carry Greedy routing if needed, and l and a are small constants. So the overall forwarding complexity is only $O(\log n)$. In most cases, Greedy routing is not needed and the forwarding decision of RASTER is much more efficient than that of Greedy routing. Also, it is sufficient for nodes to exchange bitmaps at the highest maintained resolution r_{max} and then nodes would use these to accurately construct low resolution bitmaps. The cost of this *exchange* is $\Theta(a \cdot r_{max} \cdot \log n)$ bits, and the *storage cost* is $\Theta(a \cdot l \cdot r_{max} \cdot \log n)$ bits.

In comparison, if we were to extend the NoN routing by storing routing tables within a radius of a hops, the forwarding decision overhead would become $\Theta(\log^a n)$ operations (typically on 160-bit numbers), which is proportional to the number of routing table entries maintained. The overhead involved in exchanging routing table information and in storing them would be $O(c \cdot \log^a n)$ bits, where c is the size of each entry, usually assigned 160 bits. Figure 3.1 and Table 3.2 compare the protocol complexity of NoN and RASTER.

It is clear from the figures that the forwarding overhead of RASTER is much more efficient that of NoN. Also, while the propagation and storage overheads of RASTER and NoN are comparable when $a = 2$, RASTER outperforms NoN when $a > 2$.

3.2.5 Impact of Bitmaps' Resolution

Careful inspection of the RASTER routing algorithm reveals two phases: inter-bin and intra-bin. In the inter-bin phase, a query is forwarded using bitmaps toward any node in the destination bin; in the intra-bin phase, a query is greedily forwarded within the

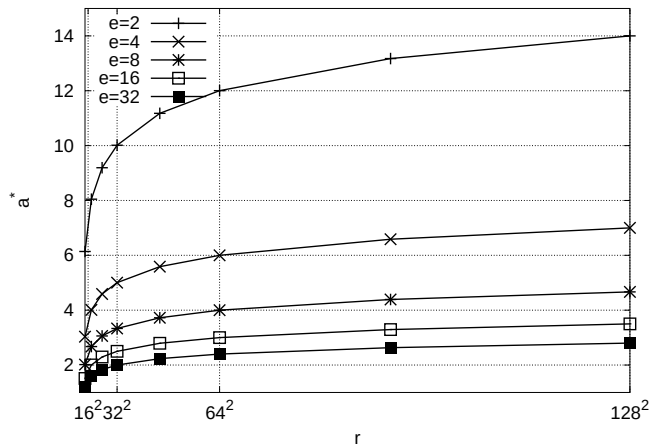

Figure 3.3: Optimal value, a^*, of the coverage radius a as the resolution r and the average number of random connections e vary.

destination bin toward the destination node. Let H_1 and H_2 be the average lookup path length in the intra-bin and inter-bin phases. The question is, in a randomized DHT of n nodes in which each node maintains on average e random connections, what is the optimal resolution, r^*, and the optimal coverage radius, a^*, to minimize the overall average lookup path length, $H = H_1 + H_2$? Since the larger r and a the larger the overhead, it is desirable to minimize r and a without sacrificing performance.

Let n_1 be the number of nodes in the destination bin and e_1 be the average number of random connections from a node to other nodes in the same bin. Using a resolution of r to construct bitmaps leads to $n_1 = n/r$ and $e_1 = e(n_1/n) = e/r$. Typically $e \ll r$ and thus $e_1 \approx 0$ and the impact of e_1 on the intra-bin routing is negligible. H_1 can be approximated as the average path length in a network of size n_1 with no random connections, which is $0.5\sqrt{n_1}$ in the case of a 2-dimensional torus[43].

During the inter-bin routing phase, cross-bin long hops mostly follow random connections. This phase can be approximated as a random graph of size $n_2 = r$ *virtual nodes* that are interconnected with the nodes' random connections. Let e_2 be the average number of random connections of a node that point to nodes in other bins. Since $e \ll r$, $e_2 = e - e_1 \approx e$. The optimal inter-bin lookup path length in such a random graph is equal to the diameter of the graph, which is approximately $\log_e r$. This optimal performance can

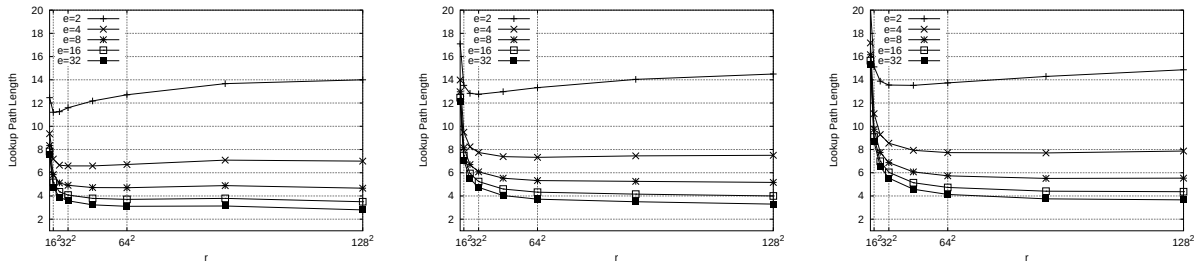

Figure 3.4: Lookup path length when the average number of random connections $e = 2, 4, 8, 16, 32$, using coverage radius a^* in a CAN system of $n = 10k$ (left), $n = 30k$ (middle), and $n = 50k$ (right).

be achieved with a coverage radius, a^*, also equal to the diameter of the graph. That is,

$$a^* = \log_e r \tag{3.1}$$

In this case, the overall lookup path length is,

$$H \approx 0.5\sqrt{n/r} + \log_e r \tag{3.2}$$

An important observation from this analysis is that a^* is independent of n but dependent on r and e. Figure 3.3 plots the values of a^* for different values of r and e. It is clear from the figures that (1) a^* is more sensitive to e than to the choice of r especially for reasonably large values of r. (2) For $e > 2$, a^* almost saturates at $r = 64^2$ and the change in a^* for larger r is logarithmic. As a result, the marginal utility from increasing r above 64^2 does not justify the inherent cost in dealing with bitmaps of larger resolution. The figure also reveals that $a = 2$ is only optimal for $e \geq 32$, which however is rarely the case for a randomized DHT because of its capability of achieving a small network diameter $\Theta(log_k n)$ with a small node degree k. This motivates the need for larger coverage radius $a > 2$ for $e < 32$. In Figure 3.4 we plot the average lookup path length, H, for different values of r and e and when using the appropriate a^*, for $n = 10k$ (left), $n = 30k$ (middle), and $n = 50k$ (right). The main message from these figures is that H is not sensitive to an increase in n/r of less than an order of magnitude, which can be explained by that n only affects $0.5\sqrt{n/r}$ of the value of H. However, the above analysis is based on a worst-case scenario that random

connections barely connect nodes within the same bin. For $n >> r$, we can greatly improve the routing performance without increasing r proportional to n by allocating a fraction of e to intra-bin connections only, which gives $H \approx \log_{e1} \frac{n}{r} + \log_{e2} r$, where $e_1 + e_2 = e$. This implies that RASTER would perform even better in small-world networks for extremely large n.

3.3 Performance Evaluation

In this section, we study through simulations the performance of the following three routing strategies: (1) Greedy, (2) NoN, and (3) RASTER. We compare the performance of these strategies to the optimal shortest-path (SP) performance, which can be achieved by constructing Shortest-Path Spanning Trees across the whole system. While not practical, SP sets the bar for other routing strategies.

3.3.1 Performance Metrics

In our simulations, we use the following metrics to evaluate the performance of the routing protocols.

- *lookup path length.* Is the average number of overlay hops needed by a routing strategy to route a message between random pairs of nodes.
- *lookup success rate.* Is the fraction of queries that are successfully delivered to their destinations. Query delivery may fail in the presence of incomplete/inaccurate routing tables due to system churn.

3.3.2 Simulation Setup

We have implemented the routing protocols on a randomized 2-dimensional CAN [43] of $30K$ nodes, in which nodes maintain connections to their CAN neighbors and to some arbitrary nodes at random. We distributed the number of random connections between nodes by allowing each node to connect to 10 random neighbors. The readers can refer to [43] for details about CAN overlay construction. Each experiment was conducted 30 times using different random connections; and in each case, we select a thousand random

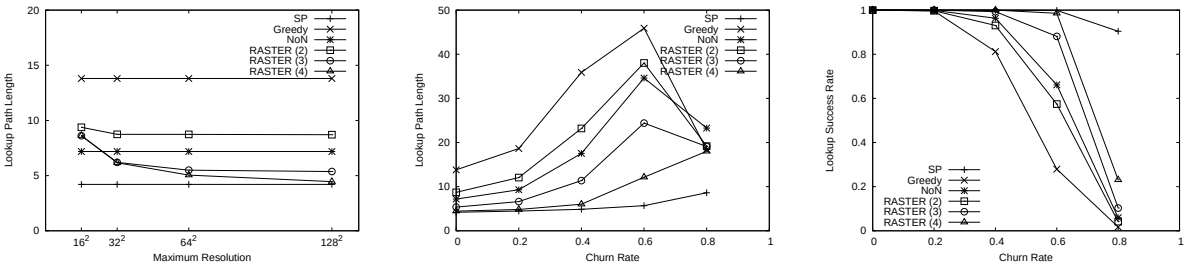

Figure 3.5: Lookup path length of RASTER(a) compared to other routing protocols as we vary the maximum resolution r_{max} (left). Lookup path length (middle) and lookup success rate (right) of RASTER with $r_{max} = 64^2$ under the impact of system churn in comparison to other routing strategies when nodes are unaware of up-to 30% of other nodes' departures.

source-destination pairs to test each of Greedy, NoN, and RASTER routing protocols and the results were averaged. Our implementation of RASTER in this section only serves as a proof of concept. In general, the protocol can be implemented on an arbitrary DHT.

3.3.3 Experimental Results

Our experiments were targeted at investigating two issues: (1) The impact of the maximum bitmap resolution r_{max} on the performance of the RASTER protocol, and (2) compare the performance of the different routing protocols under churn. That is when a fraction of the nodes have left the system possibly without informing their neighbors and the routing tables information may not be quite accurate.

To study the impact of r_{max}, we tested the performance in terms of the lookup path length when r_{max}=16^2, 32^2, 64^2, and 128^2. In each case, we set the number of maintained resolutions based on r_{max}. That is, when using $r_{max} = 128^2$, we maintain the other three lower resolutions 16^2, 32^2, and 64^2; when using $r_{max} = 64^2$, we maintain the other two lower resolutions; etc. As shown in Figure 3.5-(left), Greedy routing performs the worst (13.8 hops), while the optimal SP performance is only 4.2 hops, and NoN stands at about 7.8 hops. As r_{max} or the number of considered neighbors increases, the performance of the RASTER protocol becomes more and more evident. RASTER(4) hits the optimal SP performance when using r_{max} of 128^2. Even for smaller r_{max}, RASTER(3) and RASTER(4) still outperform NoN. Given the better efficiency in maintaining, exchanging, and executing forwarding decisions when dealing with bitmaps, as outlined in Figure 3.2, RASTER's gain

in performance over maintaining full routing tables almost comes for free; better yet, with a reduction in cost. The experimental results of RASTER also match our analysis on a $30k$-node case in Section 3.2.5.

In order to compare the performance of the different routing protocols under churn, we dropped off some fractions of the nodes from the system: .2, .4, .6, and .8. We refer to the fraction of dropped nodes as the *churn rate*. For each case, we keep the overlay neighbors informed about 70% of the dropped nodes, which in reality will happen through a connection timeout mechanism, while 30% of the dropped nodes may clutter routing information with inconsistencies. Our choice of 70% is arbitrary but different values do not change our conclusions. This inaccurate information can lead to cases where queries fail to reach their intended destination. Also, for this experiment, we use a maximum resolution r_{max} of 64^2.

In Figure 3.5 we plot the lookup path length (middle) and the lookup success rate (right) as the churn rate increases. The figure shows that Greedy routing reacts poorly to churn; although NoN and RASTER(2) outperform Greedy, the gap between the optimal performance and theirs, in terms of both the lookup path length and success rate, increases as the system churn grows. It is also clear from the figure that extending nodes' routing views to 3 or 4 hops (using RASTER(3) or RASTER(4)) does reduce this performance gap. Since extending nodes' views by storing full routing tables is intolerable, as outlined in Figure 3.2, encoding routing tables as bitmaps and using a routing protocol like RASTER can help mediate the performance degradation due to churn, which is inherent in P2P systems. This can be explained by that, in the presence of churn, some next-hop choices that appear valid to Greedy or NoN may actually lead to dead-end paths. However, as we extend nodes' view further, RASTER routing can easily detect most of these dead-end paths at intermediate hop.

Chapter 4

On the Stability-Scalability Tradeoff of DHT Deployment

4.1 Introduction

DHTs typically serve as shared storage infrastructures where stored data objects are mapped to a global identifier space. By partitioning the identifier space across a set of distributed nodes and connecting them into an overlay network, DHTs provide efficient data naming and location with simple hash-table-like primitives, upon which sophisticated distributed applications such as storage [15], content distribution [12] and more can be built.

In order to ensure an adequate level of service availability and performance, a DHT needs to monitor and maintain the connectivity of its overlay and the availability of its stored content. This maintenance requires DHT nodes to constantly probe other nodes, to replicate/code/migrate content, which consumes a large amount of the system resources especially in inherently dynamic P2Psystems [50, 22]. Thus, while allowing every member of a P2P system to contribute by serving leads to a wealth of service resources, it comes at the cost of high maintenance overhead consumed to recuperate for unstable nodes that spend a small amount of time in the system. The gain does not always outweigh the cost. On the other hand, relying purely on dedicated, high capacity clusters of servers to accommodate

the huge demands of P2P applications is financially costly and is not always, if at all, feasible. As a compromise between pure P2P and pure client/server paradigms, the OpenDHT model relies on a limited set of dedicated, relatively stable distributed nodes under centralized administration, and offers its services to other outsider nodes, which act merely as clients to the DHT [45]. Still, the cost of bringing up a large set of dedicated distributed nodes to meet the ever-increasing demands of P2P applications is a heavy burden.

In this chapter, we advocate the wise deployment of free, potentially unstable resources of P2P users. In other words, P2P members are welcomed to join the DHT infrastructure as long as the gain in service scalability outweighs the maintenance cost incurred due to node instability, whereas members for which the cost outweighs the gain should act only as clients. The service demands that are beyond the DHT's capacity can be handled in an application-dependent manner, be left out to dedicated application servers if available, queued and re-processed, dropped, etc. It is clear that many system attributes are beyond a system administrator's control such as the workload imposed by application users, the capacity of nodes and the time they spend in the system, the popularity of the stored data objects and the diversity of these attributes among the P2P system members. Still, an administrator/designer may be able to control or at least stimulate changes to the set of nodes serving as DHT members, the data objects to be maintained by the DHT, and the appropriate replication/coding strategy to store these objects.[1] In order to maximize the utility of the underlying system, the DHT should (1) include the most stable nodes and (2) maintain the most popular data objects. The most stable nodes are indeed needed to increase the reliability of DHT data service and to reduce the overhead associated with overlay and data maintenance, and more popular data objects are more welcomed to the DHT since they are targeted by most of user requests and answering more user requests ensures higher system utility. A careful design allows the DHT to answer as many of the user requests as possible given the uncontrolled system attributes. Figure 4.1 provides a schematic description of the propose service paradigm.

We cast the problem as a multivariate constrained optimization targeted at optimizing the DHT throughput, and use the model to show how to effectively administer DHT systems by tuning: (1) DHT node selection process, (2) DHT content selection process, (3) content redundancy strategy.

[1]The techniques to monitor and induce changes to the system are out of the scope of this dissertation.

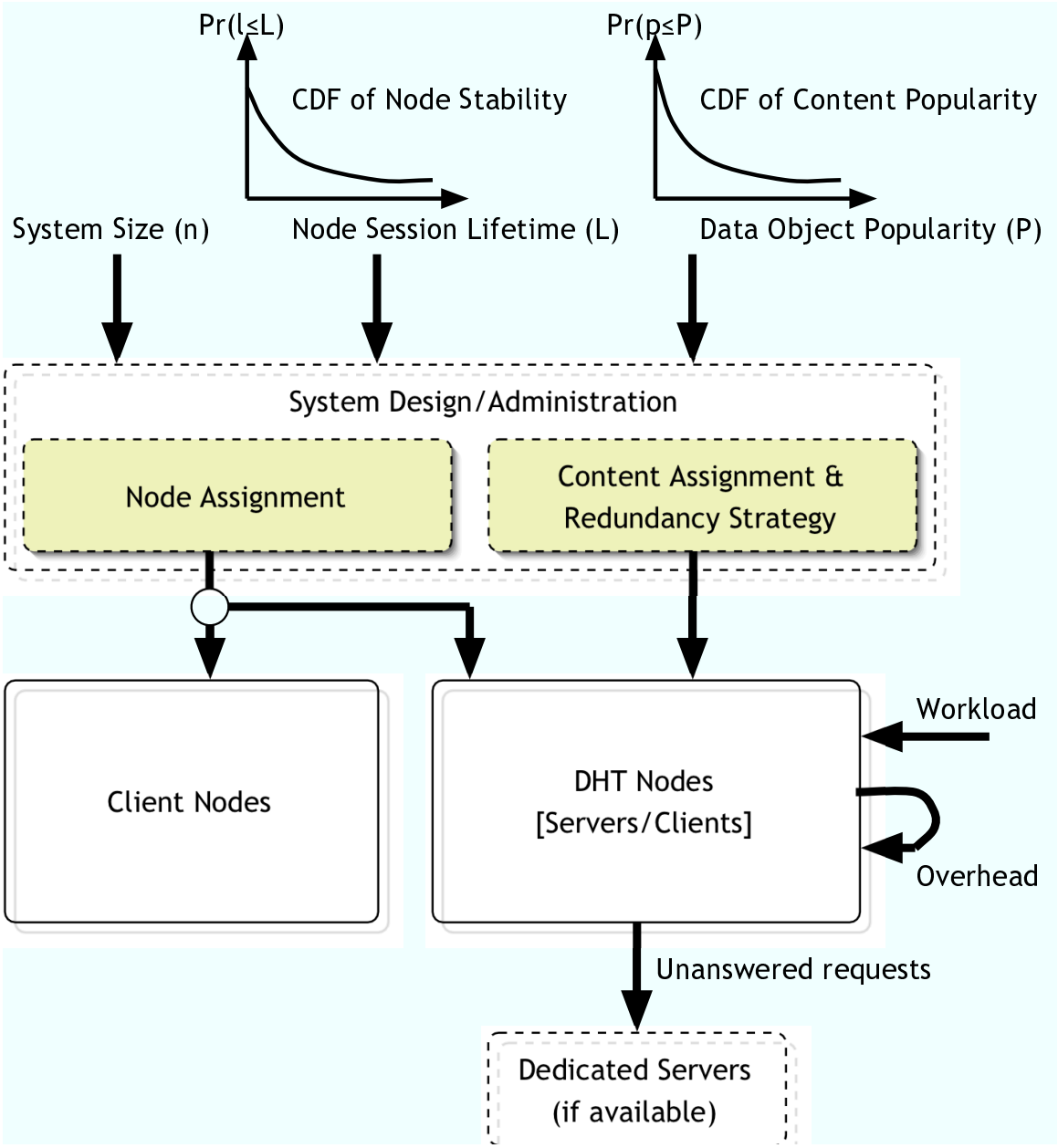

Figure 4.1: System Overview.

The rest of this chapter is organized as follows. In Section 4.2 we survey existing research on overlay stability and data availability and relate them to our own. In Section 4.3 we introduce the basics of our model, the terminology and the assumptions that we make. In Section 4.4 we model the node selection process and its implications on DHT stability and scalability. In Section 4.5 we model the content selection process and data redundancy strategies. In Section 4.6 we quantify the overhead associated with DHT deployment. In Section 4.7 we use the results of earlier sections to formulate the optimization function. In Section 4.8, we apply the model to different case studies and analyze the results.

4.2 Related Work

In order to improve DHT functionality in the presence of high node instability, researchers have taken different approaches, whether by (1) improving node stability through a selection of the most stable nodes as DHT members, or by (2) improving the availability of the maintained data objects through replication and erasure coding.

OpenDHT [45, 5] runs a DHT on PlanetLab [6] nodes, which offer services to nodes outside the DHT. PlanetLab is a wide-area testbed for networking and distributed systems research, and its nodes are much more stable than typical P2P nodes. In [21], Godfrey et al. investigate several strategies to select stable DHT nodes and explore the performance implications. Also, in [36] Mickens et al. use statistics about nodes' history to predict their future behavior and use more stable nodes for data placement. However, how stable should nodes be to join the DHT? If the process is very selective, then a small number of stable nodes will qualify, which may stifle the system scalability and limit its ability to respond to heavy workloads. If the process is not selective, then less stable nodes will qualify and the associated overhead will kick-in. None of the previous approaches have studied this tradeoff to determine the optimal stability threshold, which is a motivation of our work.

In [10], Black et al. show that when nodes are not stable, high data redundancy requires unbearable cross-system bandwidth. In [48], Rodrigues et al. compare redundancy schemes for DHTs, taking overlay node characteristics into account. They conclude that erasure coding favors environments with low node stability but the required maintenance bandwidth for a scalable and highly available system can be unsustainable for home users. In [54], Weatherspoon et al. estimate the amount of data redundancy required for reducing

data recovery costs incurred by transient node failures. These studies did not optimize the utility of the system as reflected on the number of answered queries, and did not evaluate the choice of the redundancy schemes under various factors such as the bandwidth and storage constraints, the skew in the popularity of the maintained objects, and the workload imposed on the system, which is a motivation of this work.

4.3 Model Overview

Table 4.1: Terminology for DHT Model.

Symbol	Meaning
n	number of nodes in the system
a	node availability of the system
a'	node availability of the DHT set
q_d	mean size of data objects
q_m	mean size of DHT message
σ_r	per-node READ rate
γ	ratio of per-node WRITE rate to READ rate
c_b	mean bandwidth budget per DHT node
c_s	mean storage budget per DHT node
W	number of data objects to be stored in the DHT
l	number of data fragments generated per WRITE
m	number of data fragments needed per READ
r	rate of coding
A	availability of DHT data object
$U(x)$	CDF of system nodes' session time
$u'(x)$	CDF of the DHT nodes' session time
$\overline{u}$	mean session time of system nodes
$\overline{d}$	mean downtime of system nodes
$\overline{u'}$	mean session time of the DHT set

Our model is targeted to simple storage systems in which *throughput*, the amount of DHT data that can be retrieved by user requests in some time period, is the performance metric to be optimized. Our objective is mainly to offer qualitative insights into the problem rather than accurate quantitative results. Resources such as bandwidth and storage capacity impose system constraints; other resources such as processing capacity are relatively abundant.

We consider a system of n nodes in the steady-state, in which short-term node dynamics like transient slowness at DHT nodes [44] have negligible transient effects on the overall system's performance. We also dismiss packet loss and retransmissions from our model. Each node cycles between two states: *ON* when the node is in its *session time* (or *uptime*); and *OFF* when in its *downtime*. A node's session time is a time interval from the moment it joins the system to the moment it subsequently departs, while its downtime is a time interval from its departure to its subsequent re-join. Let $U(.)$ be the cumulative distribution function (CDF) followed by nodes' session times and $\overline{u}$ be the mean session time. The larger $\overline{u}$, the more stable the system. Similarly, let $D(.)$ be the CDF followed by nodes' downtimes and $\overline{d}$ be the mean downtime. The actual distributions of these system attributes will be presented in later sections. We define *node availability*, a, as the probability that a node is in the ON state. Thus, $a = \bar{u}/(\bar{u} + \bar{d})$, and the expected number of nodes in the ON state can be expressed as $a \cdot n$. It should be clear that a larger a does not necessarily mean a more stable system.

Nodes that are in the ON state generate *workload* in the form of READ and WRITE requests on data objects. WRITEs are used to insert new data objects at the appropriate nodes in the DHT and READs are used to retrieve these objects. Let σ_r be the average number of READ requests per unit time generated by each node in the ON state, and $\sigma_w = \gamma\sigma_r$, $0 < \gamma \leq 1$, be the average number of WRITE requests per unit time generated by each node in the ON state. Each READ and each WRITE request refers to a single data object. Let q_d be the average size in bytes of the data objects and q_m be the average size in bytes of the control messages exchanged in the system. We assume UDP-based transport protocol is used and, for convenience, assume that each DHT message has $q_m = 50$ bytes long (28 bytes for the UDP/IP header, 20 bytes for the DHT identifier, and 2 bytes for miscellaneous overhead.) [5, 47].

Our model also accounts for the overhead associated with the transfer of data objects. Assuming 1500 byte packets, an object of size q_d bytes will fill out the payload of

approximately $\lceil \frac{q_d}{1500-q_m} \rceil$ packets, each of which consumes q_m bytes of overhead.

To avoid storing an infinite number of data objects in the DHT, each object is purged out of the DHT after some time period, τ, but the purged objects could be re-inserted as nodes wish. In steady state, the average number of unique objects in the system, W, can be expressed as $W \propto an\sigma_w\tau$. We assume objects' popularity follows some distribution $P(.)$, which we investigate in detail in Section 4.5. To keep our analysis manageable, we assume that DHT nodes do not lose references to their assigned data objects if they leave then rejoin the system as long as the objects time-to-live (TTL) value, τ, does not expire [10, 48, 54].

Nodes are either assigned to the *DHT set* or the *client set*. Nodes in the DHT set participate in serving client requests whether generated by nodes in the client set or in the DHT set, and nodes in the client set are pure clients. Assuming enough information about nodes' session times [21, 36], nodes with expected session times above some threshold, T, are assigned to the DHT when they are in the ON state. DHT nodes that switch to the OFF state and temporarily depart the DHT, do not respond to clients' requests. In order to avoid losing content to nodes in the OFF state, data redundancy schemes such as replication and erasure coding are used. Clients' requests are forwarded to appropriate DHT nodes which maintain references to DHT nodes storing the actual data objects and these references may be outdated due to DHT nodes' temporary departure. This transient departure model of DHT nodes has been used and justified in DHT research [10, 48, 54]. Let $n' \leq n$ be the number nodes in the ON state that are assigned to the DHT set and $\eta \equiv n'/(a \cdot n)$ be the fraction of such nodes, $0 < \eta \leq 1$. The most stable nodes are assigned to the DHT. Also, let $w' \leq W$ be the number of data objects (files) managed by the DHT and $\xi = w'/W$ be the fraction of such objects, $0 < \xi \leq 1$. The most popular files are managed by the DHT. The choice of η and ξ implies stability, availability, capacity DHT characteristics. Let $\overline{u'}$, $\overline{d'}$, $u'(.)$, $D'(.)$, a' denote the attributes describing the DHT set, corresponding to the $\overline{u}$, $\overline{d}$, $U(.)$, $D(.)$ and a attributes describing the pool of n nodes, respectively. Furthermore, we define $\check{a}'$ as the probability that a DHT node is in the ON state and stays ON long enough to serve a client request.

We assume a generic DHT structure in which each DHT node maintains overlay connections to a set of overlay neighbors and $\log_2(n')$ successor nodes [41, 51]. The expected total DHT bandwidth and storage budget available to the DHT system are $n' \cdot c_b$ and $n' \cdot c_s$, respectively. We assume that the DHT workload is balanced among its nodes by an underlying load balancing mechanism [19, 20]. Typically, the load balancing procedure

incurs periodic data transfer in DHTs, but we simply assume that the cost of this traffic is incorporated in the value of σ_w. The symbols used throughout this chapter are summarized in Table 4.1.

4.4 Node Assignment

In this section we rely on P2P measurement results to pinpoint realistic node session time and downtime distributions, $U(.)$ and $D(.)$, respectively. Then show how we select $n' < n$ nodes in the ON state, a fraction $\eta = n'/(a \cdot n)$, as DHT nodes and study the impact of our node selection on the DHT system stability, $\overline{u'}$, and node availability, a' and $\check{a}'$.

4.4.1 Distribution of Node Session Times

We assume apriori knowledge about nodes' expected session times, which can be obtained by directly questioning the nodes, by monitoring nodes' session-time history and using it as an indication of their future behavior [21, 36]. Measurement results indicate that nodes' session times can be well modeled by a long-tailed distribution [50, 54, 31]. We assume that nodes' expected session times are independently distributed and model them using a shifted Pareto with probability density function (PDF)

$$u(t) = \frac{\alpha}{\beta}\left(1 + \frac{t}{\beta}\right)^{-(\alpha+1)}, \ \alpha > 1 \tag{4.1}$$

and cumulative distribution function (CDF)

$$U(t) = 1 - \left(1 + \frac{t}{\beta}\right)^{-\alpha}, \ \alpha > 1 \tag{4.2}$$

The smaller the value of α the more stable the system as nodes stay longer in the ON state. We are not aware of any P2P measurement study that reveals the distribution of node downtimes. We assume that the average downtime, $\overline{d'}$, is a constant and equal to 5.25 hours according to [9].

4.4.2 DHT Stability

Nodes in the ON state, which are expected to stay alive for at least some time, T, are included in the DHT set; otherwise, they are included in the client set. The PDF of DHT nodes' session times, $u'(.)$, and the CDF of DHT nodes' session times, $u'(.)$, can thus be expressed as

$$u'(t) = \begin{cases} 0 & t < T \\ K_T u(t) & t \geq T, \end{cases} \tag{4.3}$$

where K_T is a normalization constant and can be expressed as $K_T = 1/\int_{t=T}^{\infty} u(t)dt = 1/(1 - U(T)) = (1 + T/\beta)^{\alpha}$, and

$$u'(t) = \begin{cases} 0 & t < T \\ K_T(U(t) - U(T)) & t \geq T, \end{cases} \tag{4.4}$$

Using the average session time of DHT nodes, $\overline{u'}$, as a measure of DHT stability, it can be expressed as: $\overline{u'} = \int_{t=T}^{\infty} t \cdot u'(t)dt = K_T \int_{t=T}^{\infty} t \cdot u(t)dt$. Substituting the value of K_T and $u(t)$ from Equation 4.1, and solving the integration, we get

$$\overline{u'} = \frac{\beta + \alpha T}{\alpha - 1} \tag{4.5}$$

The larger the threshold, T, the more stable the DHT system becomes (larger $\overline{u'}$) since only the most stable nodes join the DHT.

4.4.3 DHT Scalability

On the other hand, the larger the threshold, T, the less scalable the DHT system (smaller n' and η) since the bandwidth budget offered by the small number of DHT nodes is small.

We next identify the values of n' and η (scalability measures) as a function of the DHT node selection threshold, T. Using Little's theorem, the average number of nodes in a stable system is equal to the average arrival rate of these nodes multiplied by their average

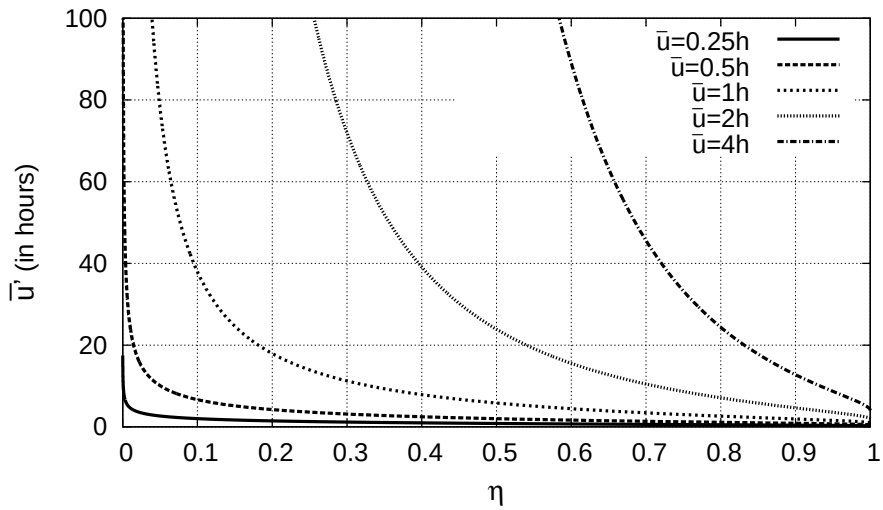

Figure 4.2: Scalability (η) vs. Stability ($\overline{u}'$) for different values of the node selection threshold, T.

lifetime in the system. Applying Little's law to nodes in the ON state in our system, we get

$$an = \lambda \overline{u} \tag{4.6}$$

where λ is the average arrival rate of nodes to the ON state. Applying Little's law to nodes in the ON state, with expected lifetime larger than T (the DHT nodes), we get

$$n' = \lambda(1 - U(T))\overline{u'} \tag{4.7}$$

Solving equations 4.6 and 4.7, and substituting for $U(T)$ from Equation 4.2, and using $\overline{u} = \int_{t=0}^{\infty} t \cdot u(t)dt = \beta/(\alpha - 1)$ then

$$\eta \equiv \frac{n'}{an} = \left(1 + \frac{T}{\beta}\right)^{-\alpha} \left(1 + \frac{\alpha}{\beta}T\right) \tag{4.8}$$

Based on equations 4.5 and 4.8, the rate of change of $\overline{u'}$ as T changes can be expressed as $\partial\overline{u'}/\partial T = \alpha/\alpha - 1$, while the rate of change of η as T changes, setting $\beta = 1$ for simplicity, can be expressed as $\partial\eta/\partial T = -\alpha(\alpha - 1)T/(1 + T)^{\alpha+1}$. This means that increasing T

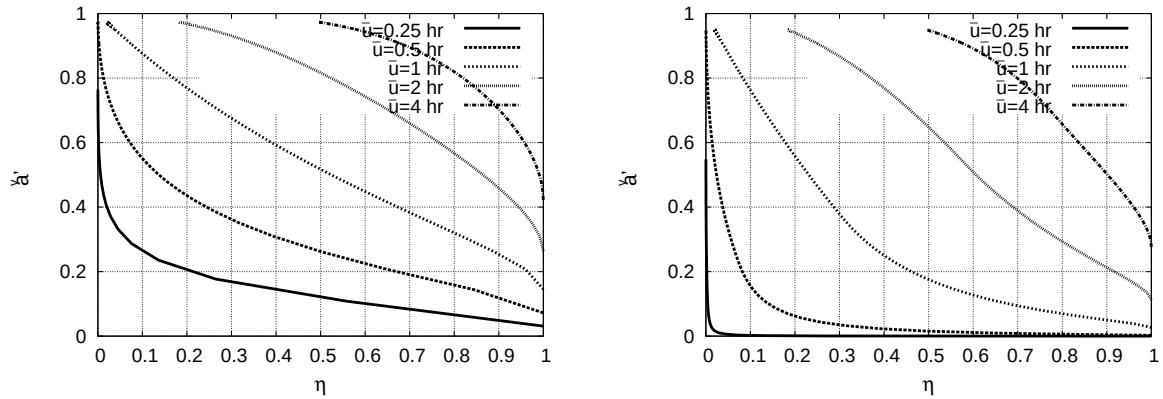

Figure 4.3: DHT node availability, $\check{a}'$, as η increases when average sojourn time, $\overline{s}$, is 0.1 (left) and 5.0 (right).

degrades the system scalability at a higher pace than the pace at which the system stability is improved. To get some intuition into the degradation in stability, $\overline{u'}$, as scalability, η, increases refer to Figure 4.2. We vary α between 5, 3, 2, 1.5, and 1.25 to obtain $\overline{u}$ of 0.25, 0.5, 1, 2, and 4 hours, respectively. Then for each of these cases, we plot the curve of $\overline{u'}$ as η changes. Clearly, DHT stability drops dramatically as more nodes join the DHT. This calls for accurate tuning of the node selection process as the degradation in DHT stability can easily degrade the system performance.

4.4.4 DHT Node Availability

Recall that a client request is answered by a list of references to nodes storing the requested content. The referenced nodes may have left the system. A reference to a node will be successful if (1) the node is in the ON state, and (2) if it stays long enough in the ON state to serve the requested content to the client.[2] The probability that a DHT node is in the ON state, a', can be expressed as

$$a' = \frac{\overline{u'}}{\overline{u'} + \overline{d'}} \tag{4.9}$$

Let $\check{a}'$ be the probability that a DHT node is in the ON state *and* stays ON long

[2]We assume that a client does not leave the system while downloading its requested content.

enough to answer a client request. In order to estimate $\check{a}'$, we define two measures: (1) a DHT node's *residual lifetime*, which is defined as the amount of time remaining in the node's session time at the moment a client request is intercepted. (2) A request's *sojourn time* is the amount of time that a client's request is expected to spend at a DHT node for the requested content transfer, until the request is fully served. The sojourn time includes the queuing time and the service time of the request. The following distributions are of special interest to the estimation of $\check{a}'$: (1) the PDF of DHT nodes' residual lifetimes, $r(t)$, and (2) the CDF of requests' sojourn times, $S(t)$.

Assuming requests arrive at DHT nodes uniformly at random over their session time. Then by renewal theory [27]

$$r(t) = \frac{1 - U'(t)}{\overline{u'}} \tag{4.10}$$

Also, assuming that data objects are of the same size, and the service (transmission) time of each data object is b time units, then each DHT node can be modeled as a $M/D/1$ queue with infinite buffer. Then $S(t)$ is given by [46]

$$S(t) = \begin{cases} 0 & t < b \\ (1-\rho)\sum_{k=0}^{\lfloor y \rfloor} \frac{(\rho(k-y))^k}{k!} e^{-\rho(k-y)} & t \geq b \end{cases} \tag{4.11}$$

where $y = \frac{t-b}{b}$ and ρ is the utilization of the DHT node.

Given estimates for a', $r(t)$ and $S(t)$, $\check{a}'$ can be estimated as

$$\check{a}' = a' \int_{t=0}^{\infty} S(t) r(t)\, dt \tag{4.12}$$

To get some intuition into the impact of different system parameters on DHT node availability, $\check{a}'$, we fix $\beta = 1$ and vary α as in Section 4.4.3 to obtain different node uptime distributions, $U(.)$, with $\overline{u}$ of 0.25, 0.5, 1, 2, and 4 hours. We also fix ρ and b values to obtain different sojourn time distributions, $S(.)$, with average sojourn times, $\overline{s}$, of 0.1 and 5.0 hours. The latter case is intentionally extreme to highlight the difference. In Figure 4.3 we plot the values of $\check{a}'$ as η increases for different values of $\overline{u}$ when $\overline{s} = 0.1$ (left) and when $\overline{s} = 5,0$ (right). As expected, a smaller $\overline{s}$ leads to a better chance at having DHT

nodes available in the ON state long enough to serve the requests (larger $\check{a}'$). A small $\overline{s}$ is also desirable from clients' perspective as it implies better perceived response time to their requests. In order to minimize $\overline{s}$, a system designer can either reduce the load, ρ, on DHT nodes and/or reduce b by reducing the size of the requested data objects (to reduce their transmission time) through object fragmentation as we describe in Section 4.5. Note that fragmentation slightly increases the load on the DHT system due to the extra messaging overhead. Also, the load on the system can be reduced by increasing the number of DHT nodes. This comes however at the cost of system instability. The multitude of interacting tradeoffs makes the choice of an optimal set of system parameters non-trivial and motivates the optimization that we describe in Section 4.7.

4.5 Content Assignment & Redundancy Strategy

Selecting a subset, $w' \leq W$, of the most popular data objects to be be managed by the DHT, instead of managing all W data objects, can help the system answer more of READ requests especially when stable DHT resources are limited. Data redundancy schemes are useful in offering more download points (service capacity) for popular content that would otherwise be rarely used to serve unpopular content. Data fragmentation is useful to improve the system response time and DHT nodes' availability as motivated in Section 4.4.4. In this section we rely on P2P measurement results to pinpoint realistic object popularity distributions, $P(.)$, and select a fraction $\xi = \frac{w'}{W}$ of the most popular objects to be managed by the DHT. We also survey well-known data redundancy strategies and analyze the relation between ξ and the probability that a requested data object is available for clients to download from the DHT.

4.5.1 Distribution of Object Popularity

Measurement studies indicate that the popularity of files in P2P systems exhibit a pronounced skew, with most of the file references targeted at a small number of files, and a small number of references to the majority of the files [22]. We use the widely accepted Zipf distribution with Zipf-exponent of $\alpha = 1$ to model this skew in data-object popularity,

which has a CDF of the form:

$$P(w) = \left(\frac{\sum_{i=1}^{w} i^{-\alpha}}{\sum_{i=1}^{W} i^{-\alpha}} \right) \; 1 \leq w \leq W \tag{4.13}$$

4.5.2 Data Redundancy Strategies

Past work on DHT-based storage systems [10, 16] generalize commonly adopted *replication* and *erasure coding* data redundancy schemes into a choice of coding parameters, l and m, where l is the number of data fragments generated per stored data object and $m \leq l$ is the number of fragments that need to be retrieved from the l fragments, in order to successfully reconstruct the requested data object. Replication is the special case in which $m = 1$, and l denotes the number of replicas of the data object. The *rate of coding*, $r = \frac{l}{m}$, hence expresses the amount of redundancy.

We argue that managing only a fraction $\xi = w'/W$ of the most popular data objects by the DHT, while ignoring the rest as they are beyond the capacity of the available DHT resources, and replicating the managed DHT objects can help the DHT system answer more requests. To make the case for this argument notice that (1) at most a fraction $P(w')$ of the client requests will be answered by the DHT, and (2) redundancy strategies are aimed at providing enough DHT service capacity to answer the fraction $P(w')$ of the requests. One can generalize our measure of node availability, $\check{a}'$, to incorporate the selected fraction of DHT objects, ξ, and the redundancy strategy, (r, m), as follows. Let A be the probability that a requested data object is maintained by the DHT and that m out of l fragments of the stored object are downloadable from the DHT. A can be expressed as:

$$A = P(w') \sum_{i=m}^{l} \binom{l}{i} (\check{a}')^i (1 - \check{a}')^{l-i} \tag{4.14}$$

A larger A results in better DHT throughput. We will show that using $\xi < 1$ and $r > 1$ may improve A. In Figure 4.4, we pick $\check{a}' = 0.6$ and plot A as we vary the *storage cost*, $r\xi$, for different redundancy strategies, (r, m). Note that different A values imply different system utilization. In order to fairly compare different cases, we need to mitigate the difference in system utilization and maintain similar workload on the system. Given

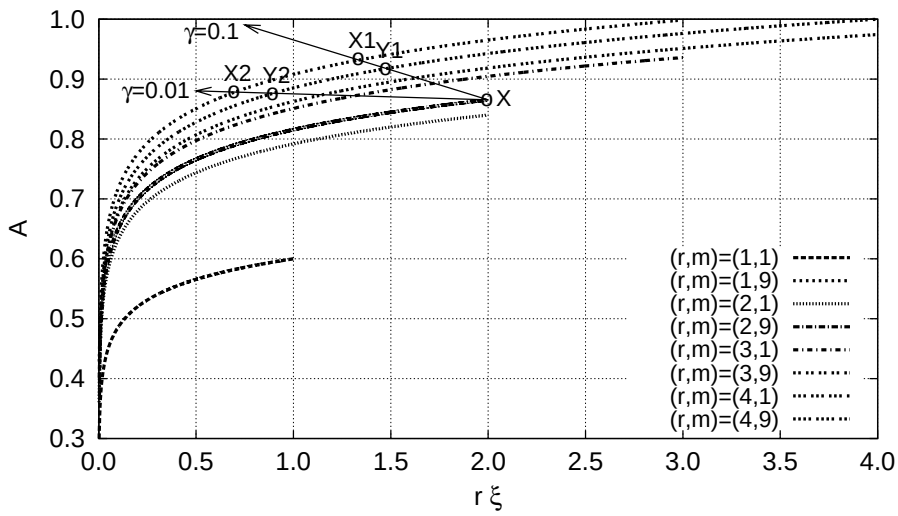

Figure 4.4: Plot of DHT throughput, A, when DHT node availability, $\check{a}' = 0.6$.

γ the ratio of per-node WRITE rate over READ rate, the workload posed on the DHT system is roughly a constant if we fix $A + \gamma r \xi$. In Figure 4.4 we plot a line labeled $\gamma = 0.1$ on which $A + \gamma r \xi$ is some fixed constant, and we pick three points X, $X1$ and $Y1$ along the line that use the same object fragmentation $m = 9$. At point X, we have $\{r = 2, m = 9, \xi = 1.0, A = 0.865\}$. At point X_1, we have $\{r = 3, m = 9, \xi = 0.447, A = 0.931\}$. Comparing X and X_1 confirms our intuition that a larger r can lead to a better A. However, at point $Y1$ we have $\{r = 4, m = 9, \xi = 0.37, A = 0.917\}$. Comparing points X_1 and Y_1 reveals that a larger r does not always lead to better A and thus an optimal value of r needs to be identified. The same conclusions hold for points X, X_2, and Y_2 along the line labeled $\gamma = 0.01$. Figure 4.4 also reveals that tuning object fragmentation, m, can improve A but marginally compared tuning the redundancy, r. A larger m typically improves the clients' perceived performance as it reduces their requests' sojourn times, but this improvement is offset by messaging overhead as we elaborate in the next sections.

4.6 DHT Resource Consumption

The maintenance of the DHT overlay structure and data, and the READ and WRITE requests all consume DHT bandwidth. The DHT data objects also consume disk storage. Our choice of η, ξ, r, and m affect the amount of consumed resources. In this

section we quantify the average resource expenditure.

The bandwidth consumed in downloading data objects by READs per online node is estimated as

$$B_1 = \sigma_r q_d A \tag{4.15}$$

B_1 provides a measure of the DHT throughput, and is the metric that our model is optimizing. The bandwidth consumed in uploading new data objects by WRITEs per online node is

$$B_2 = \gamma \sigma_r q_d r \xi \tag{4.16}$$

Each READ request and WRITE request message also consumes bandwidth. The bandwidth consumed by READ messages per online node can be expressed as

$$B_3 = \sigma_r q_m \left(H + m + \left\lceil \frac{q_d/m}{1500 - q_m} \right\rceil mP(w') \right) \tag{4.17}$$

where H is the average number of DHT hops to reach an arbitrary DHT node. The bandwidth consumed by WRITE messages per online node is

$$B_4 = \gamma \sigma_r q_m \left(H + rm + \left\lceil \frac{q_d/m}{1500 - q_m} \right\rceil rm\xi \right) \tag{4.18}$$

For maintenance traffic, we assume that each DHT node communicates periodic heartbeat messages at a rate $\frac{h}{T}$ to a set of N DHT nodes, which includes (1) the node's overlay neighbors, (2) its $\log_2(n')$ successor nodes, and (3) nodes which contain data fragments that the DHT node is referencing. We set h to 1 heartbeat message every 10 seconds, which is large enough to maintain accurate node-state information according to previous DHT studies [57, 30]. Each message incurs $\log_2(n')$ *stabilization* messages to ensure the correctness of the routing overlay [41, 51]. For $H = 1$, the DHT overlay is a full mesh so N is n'; for $H >= 2$, N is the maximum of rm and $\log_2(n') + n'^{\frac{1}{H}}$. The overall maintenance

traffic per online DHT node is,

$$B_5 = \frac{q_m N h \log_2(n')}{T} \tag{4.19}$$

The disk space consumed by all the DHT data objects can be expressed as

$$D = r w' q_d \tag{4.20}$$

Let the residual bandwidth capacity per DHT node, c'_b, be the bandwidth remaining from c_b after excluding the bandwidth consumed in DHT lookup and maintenance overhead.

$$c'_b = c_b - B_5 - \frac{\sigma_r q_m (H + m) + \gamma \sigma_r q_m (H + rm)}{\eta} \tag{4.21}$$

The service time per fragment is just a fragment's payload plus its header overhead divided by the residual bandwidth.

$$b = (q_d/m + q_m \lceil \frac{q_d/m}{1500 - q_m} \rceil)/c'_b \tag{4.22}$$

The utilization ρ is computed as the load of accepted READ/WRITE requests over the residual bandwidth (c'_b) per DHT node, which can be expressed in terms of b.

$$\rho = \frac{\sigma_r (P(w') + \gamma r \xi) m b}{\eta} \tag{4.23}$$

4.7 System Optimization

Our objective is to maximize the DHT throughput, B_1, subject to constraints on the available bandwidth and storage resources. We next formulate these constraints. The overall DHT bandwidth consumption is $an \cdot (B_1 + B_2 + B_3 + B_4) + n' \cdot B_5$ and the overall

DHT bandwidth budget is $n'c_b$, which leads to the following bandwidth constraint:

$$B_1 + B_2 + B_3 + B_4 \leq \eta \cdot (c_b - B_5)$$

Also, the storage consumed by all the DHT data objects, $rw'q_d$, must be constrained by the storage budget of the DHT, $n'c_s/a'$, which leads to the following storage constraint:

$$a' \frac{r \cdot w' \cdot q_d}{n'} \leq c_s$$

Additional constraints are needed to make our constrained optimization more realistic. First, because fragments of an object need to be placed at different DHT nodes, the constraint $rm \leq n'$ is necessary. Second, we assume that no matter how small a fragment is, $b \geq 100$ millisecond. Last, the expected sojourn time for downloading an object should be less than the average session time of nodes, so we assume $\check{a}' = 0$ for $\overline{u} < m \cdot \overline{s}$.

4.8 Model Results

The optimization problem formulated in the previous section is not amenable to closed form solution since it consists of interacting nonlinear terms, and many variables, such as n', w' and m, are bounded, positive integers. Since our objective is to offer qualitative insights and not efficient solutions, we rely on an exhaustive search of the state space (η, ξ, r, m) for the optimal combination, leading to the maximum B_1.[3]

In this section we study the evolution of the optimal DHT tuning parameters, η, ξ, r, and m, as the load on the system increases. The load exercised by each node in the ON state can be expressed as $\sigma_r q_d$ in Mega-Bytes per hour. Furthermore, we investigate the optimal tuning as (1) data object size (q_d), (2) the ratio of WRITE-to-READ workload (γ), (3) the number of system users (n), and (4) the size of data to be stored (W) varies. Towards this end, we fix the default system settings as follows unless otherwise mentioned.

[3]We limit the range of some variables to realistic values, such as $m \in [1, 100]$, and also *quantize* non-integer variables to improve the search efficiency.

We set a pool of $n = 10K$ nodes, the WRITE-to-READ ratio $\gamma = 0.01$, the average node session time $\overline{u} = 0.5$ hours, the average node downtime $\overline{d} = 5.25$ hours, the node bandwidth budget $c_b = 100MB$ per hour, the node storage budget $c_s = 10GB$, and the number of data objects $W = 10K$. In general, the observed trends persist for different setups.

4.8.1 Object Size and Throughput Performance

First we study the optimal tuning when the object size q_d is small compared to a large q_d, in order to highlight the difference between DHT applications with different object sizes. We compare scenarios with object sizes, $q_d = \{10MB, 1MB, 100KB, 10KB, 1.45KB\}$, respectively. The choices of $10MB$ and $1MB$ are to emulate the large chunk size used in file-sharing systems like bitTorrent [1]. The choice of the small $1.45KB$ object size is to contrast the large q_d's case [5] and is such that each object perfectly fits in one packet to factor out the overhead caused by packet fragmentation. Other choices of q_d are for comparison purpose. In Figure 4.5 we plot the optimal system parameters, η, ξ, r, m, the resulting DHT performance as reflected on the system throughput, B_1, the average sojourn time, $\overline{s}$, and the incurred messaging and maintenance bandwidth overhead, $B_2 + B_3 + B_4 + \eta \cdot B_5$, and the disk storage overhead, D, as we vary the load exercised by each node in the ON state on the system, $\sigma_r q_d$, between 1 to 200 MB/hour, for different q_d scenarios. The curves in Figure 4.5 reveal the following conclusions: (1) As the load on the system increases, the model reacts by incrementally including more nodes (from the most stable ones) into the DHT set, increasing η, and improving the system throughput, B_1. The act of including more nodes (improving scalability) into the DHT set continues only up-to a load value of about 60 MB/hour, when the bandwidth budget (ηc_b) is saturated, and adding more nodes does not help as the nodes that would be added are less and less stable, and would waste system bandwidth in maintenance traffic without improving B_1. η thus converges to a value even for larger load values. In scenarios of $q_d <= 1MB$, the estimate transfer time per data object at bandwidth capacity $c_b = 100$ MB/hour is relatively insignificant in comparison to the average node session time $\overline{u} = 0.5$ hours; however, in the $q_d = 10MB$ scenario, the estimate data-transfer time per object is 0.1 hours, so the DHT nodes in the $q_d = 10MB$ scenario need to be more stable than those in other small q_d scenarios for the sake of the availability of data service. This explains why in our example, η converges t a value around 0.85 for small object-size scenarios ($q_d <= 1MB$), but for the large object-size scenario

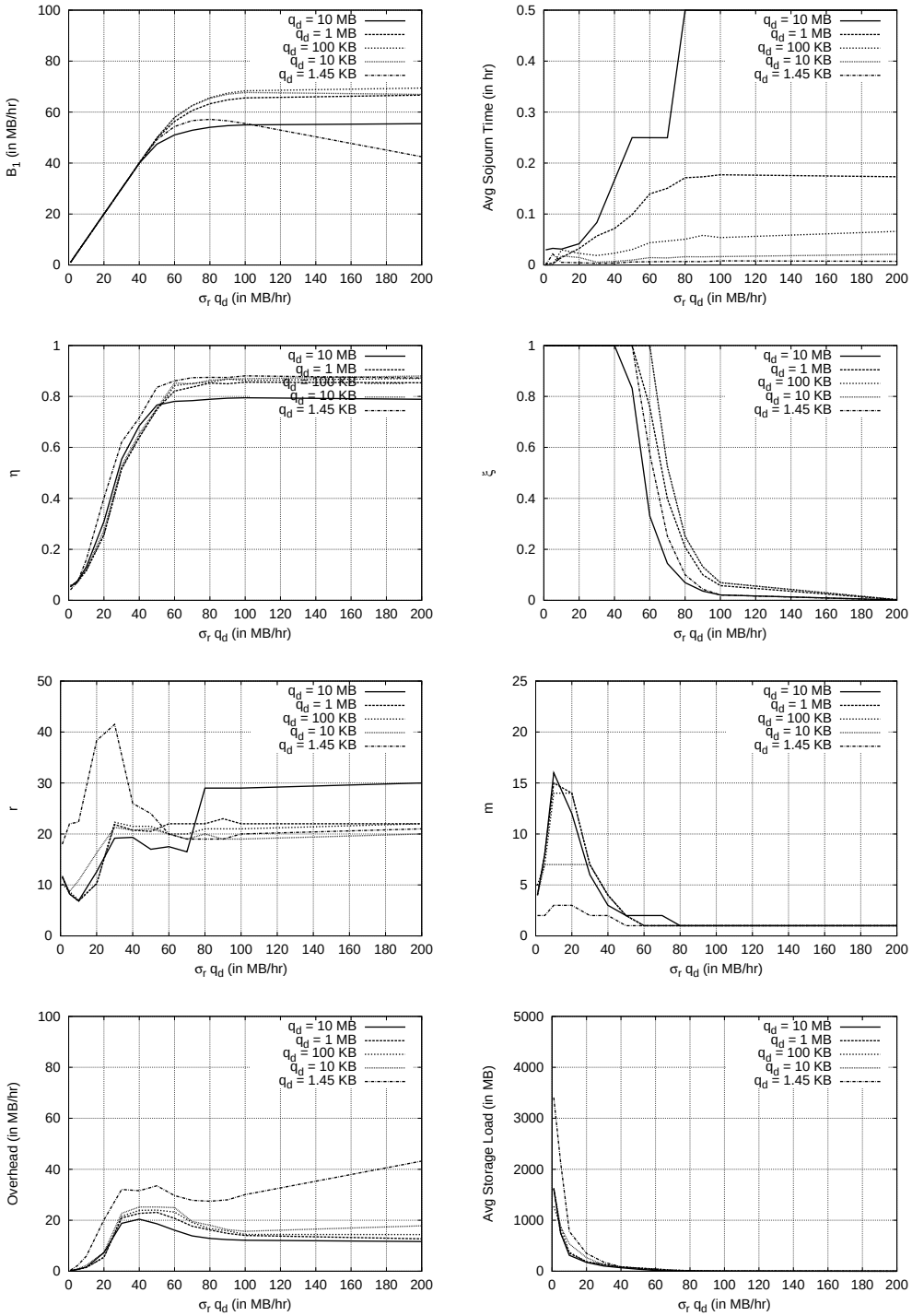

Figure 4.5: Resultant throughput of DHT model for different object size (q_d) as we vary the workload on the system.

($q_d = 10MB$), η converges to a smaller value of 0.8. (2) The fraction of objects stored in the DHT, ξ, starts to decrease once η stabilizes and the load further increases; in other words, if the system cannot gain more service capacity by adding more DHT nodes, then it needs to cut down on the number of maintained data objects, keeping only a fraction of the most popular ones in order to allocate the DHT resources more efficiently. (3) As η grows, the DHT uses high redundancy parameters r or m to counter the node instability and to maintain the availability of stored DHT objects close to 1.0. Eventually r and m stabilize as well because the the number of DHT nodes is limited and maintaining a large number of replicas, r, and fragments, m, is costly. When the load stresses the bandwidth budget, because m's improvement on A is marginal compared to r, m is reduced to 1 and the system adapts the replication strategy in order to save querying and maintenance overhead. (4) Before the workload stresses the bandwidth budget, the case of $q_d = 1.45KB$ favors the use of a higher degree of replication and therefore consumes more bandwidth and storage resources than the case of $q_d = 1MB$ without offering a better throughput, B_1. The reason for the high degree of data replication in the case of $q_d = 1.45KB$ is that $q_d = 1.45KB$ cannot exploit fragmentation effectively, since $q_d = 1.45KB$ fits in one IP packet of 1500 bytes after including the IP header. Fragmenting this packet will introduce header and messaging overhead. On the other hand, for $q_d >= 10KB$ and especially $q_d >= 100KB$, the use of a higher degree of data fragmentation is favored before the workload stresses the bandwidth budget. One interesting point is that after the workload stresses the bandwidth budget, the scenario of $q_d = 10MB$ uses a higher degree of data replication in comparison to other small object-size scenarios even though the DHT nodes in the $q_d = 10MB$ scenario is more stable than others'. Our investigation reveals that in the $q_d = 10MB$ scenario, data service at each DHT node is less available to users due to the effect of long data transfer time, so the system needs extra replicas to boost the availability of data service. (5) As η saturates, the use of $q_d = 1.45KB$ degrades the system throughput because of the heavy querying overhead, even though it leads to smaller sojourn time. The use of $q_d = 10MB$ also degrades the system throughput as η saturates mainly because due to the effect of long data transfer time, the system needs to use a smaller size of more stable nodes and a higher degree of data replication to serve the data. (6) The average sojourn time per data service in the settings of $q_d <= 1MB$ is far below the sojourn constraint of 0.5 hours, but as q_d is increased to $10MB$, the system really operates near the edge of the sojourn constraint. It shows that a system using smaller q_d can better fit in a tight sojourn constraint.

Furthermore, if the sojourn time constraint is lowered, the system can always respond by reducing ξ or increasing m to lower the average sojourn, either of which causes performance degradation.

4.8.2 System Size and Throughput Performance

DHT systems are expected to scale well with the number of system users. As the number of DHT users grows, the system must increase the number of DHT nodes in response to the workload generated by additional users. This, however, induces a larger DHT overlay network with more querying and maintenance overhead. As a result, the tendencies of the optimal settings of a large DHT system might differ from those of a small DHT system. To investigate this issue, we compare scenarios with different system sizes, $n = \{10K, 100K, 1M\}$ users. The results are plotted in Figure 4.6, which reveals the following conclusions: (1) As n increases from $10K$ to $1M$, the η curve behaves about the same before ξ starts to drop at a load value around $60MB$/hour, when the load stresses the bandwidth budget. Afterward, η converges to a value for all the scenarios except that the η convergence value drops slight from about 0.85 for $n = 10K$ to 0.8 for $n = 1M$. The optimal throughput performance, B_1, also looks the same before the load value of 60 MB/hour, after which the optimal throughput drops slightly from 65 to 60 MB/hour because of a small increase in querying and maintenance overhead as n grows. (2) The tuning of ξ also behaves in the same way as n grows. The tuning of r and m are the same except for a very small system load (less than 1 MB/hour in our example). Even trivial, the explanation is that in case of a tiny workload, a little larger η is needed by a small-size system in order to have enough DHT nodes to place $r \cdot m$ data fragments, which eventually results in some variation in the system setting. (3) As n increases, the expected sojourn time increases slightly because of the extra querying and maintenance overhead, but the increased amount of the sojourn time is insignificant in comparison to the corresponding growth rate of n. As expected, the storage load per DHT node decreases as n grows. We see that as the network size scales, the change in the system tuning is likely negligible.

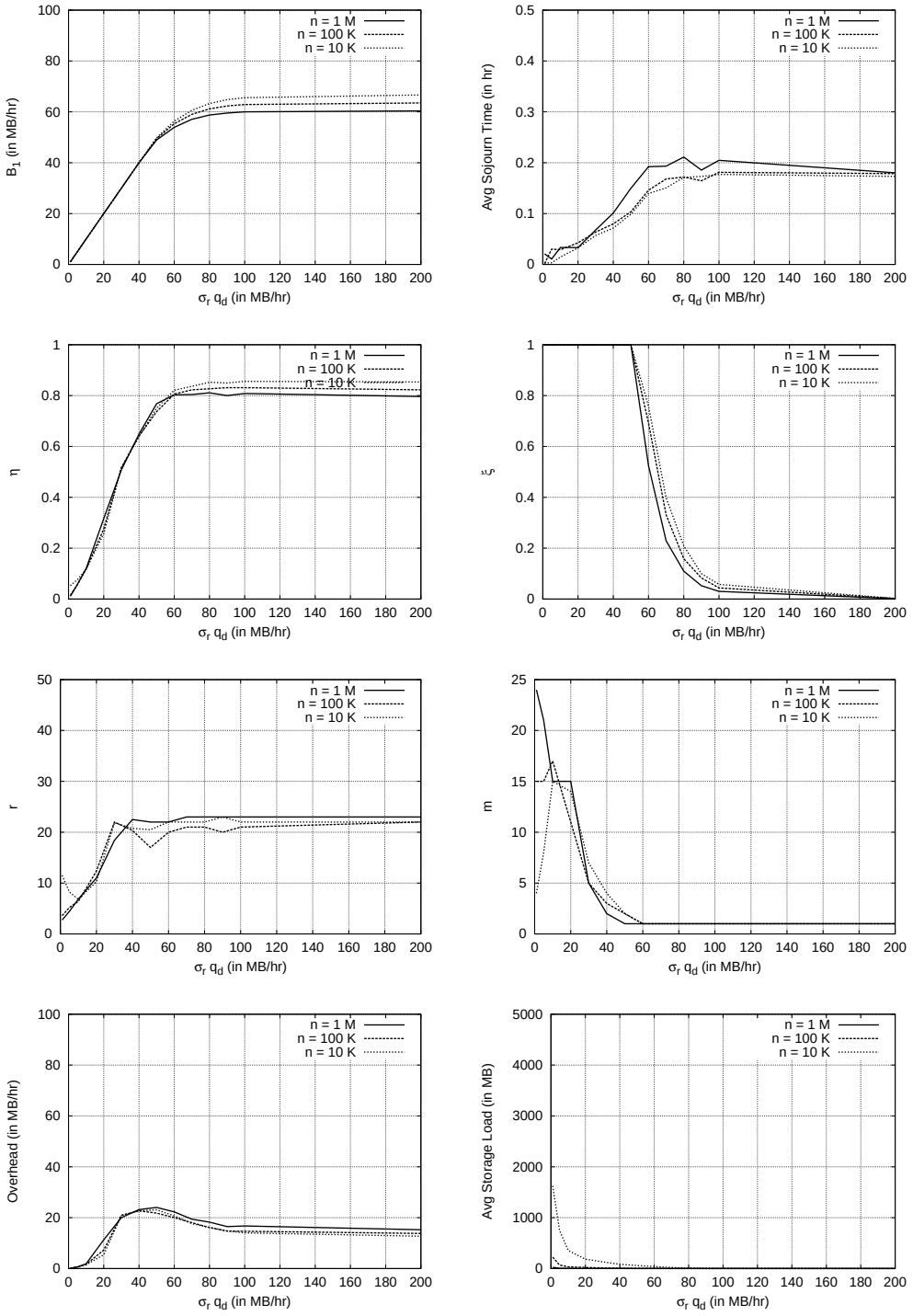

Figure 4.6: Resultant throughput of DHT model for different system size (n) as we vary the workload on the system.

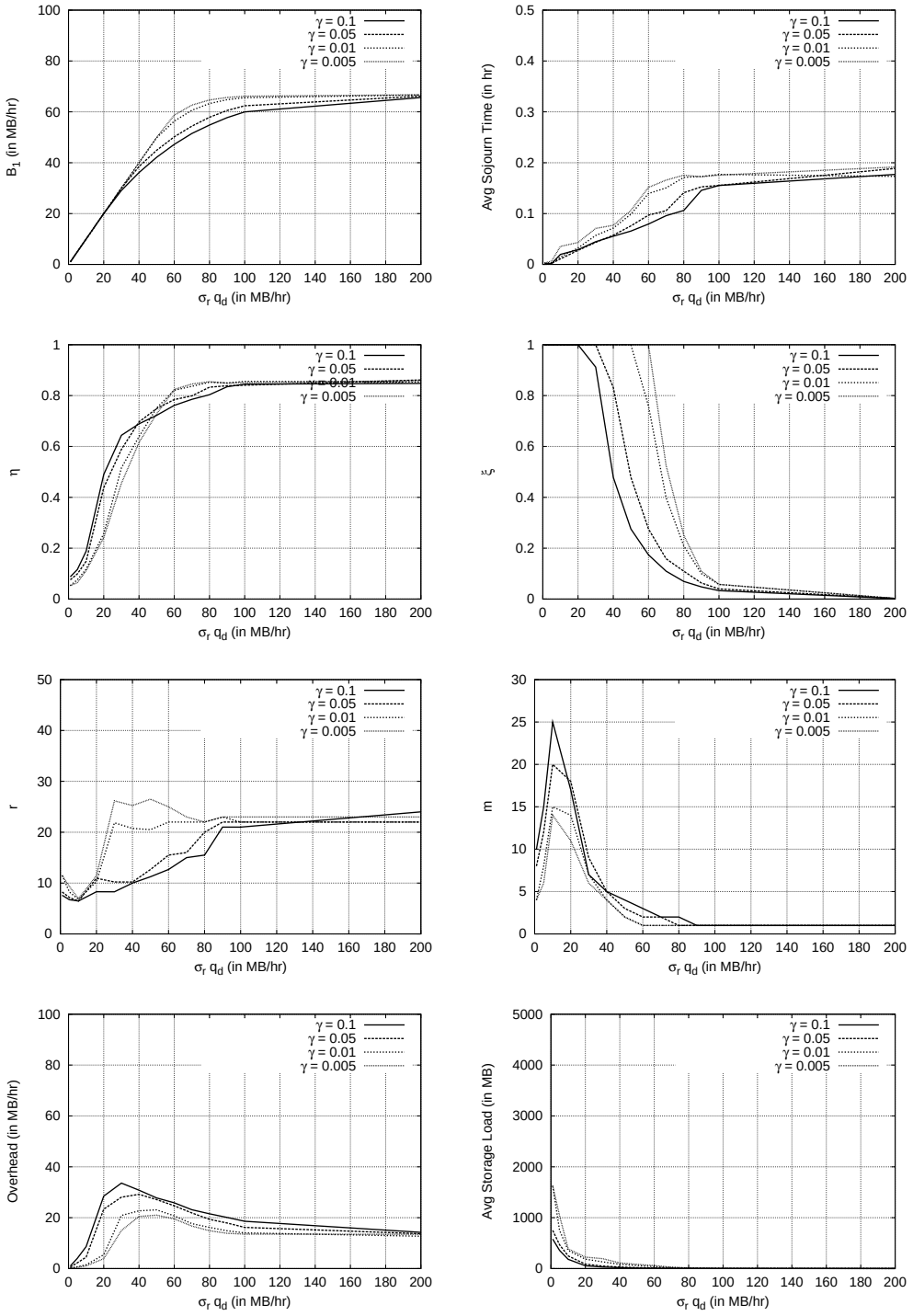

Figure 4.7: Resultant throughput of DHT model for different WRITE-to-READ ratio (γ) as we vary the workload on the system.

4.8.3 WRITE Workload and Throughput Performance

The ratio of a system's WRITE-to-READ workload to a degree affects the system's throughput performance because WRITE requests compete with READ requests for limited system resources. The WRITE workload of two DHT systems can differ from each other dramatically depending on the popularity of their stored data. For example, a highly skewed data popularity can cause a DHT system produce heavy load-balancing traffic; also, high WRITE workload is expected if the stored data needs to be constantly renewed because of their short-term popularity. To understand how the WRITE traffic affects the system's throughput performance, we compare scenarios with different WRITE-to-READ ratios, $\gamma = \{0.1, 0.05, 0.01, 0.005\}$. The results are plotted in Figure 4.7, which reveals the following conclusions: (1) In high WRITE-ratio scenarios ($\gamma = \{0.1, 0.05\}$), the system tends to use more nodes to serve (larger η) when it is underloaded because of the large WRITE workload. Furthermore, even for different WRITE ratios, η converges to about the same value as the load increases. 2) For low WRITE ratios ($\gamma = \{0.01, 0.005\}$), as the load increases, ξ begins to drop at around the point when η stabilizes as explained before. For high WRITE ratios ($\gamma = \{0.1, 0.05\}$), ξ drops much earlier before η stabilizes in order to accommodate a sufficient degree of r within the limited bandwidth budget. (3) As the load increases, high-WRITE-ratio scenarios favors fragmentation over replication to counter the node instability because the combination of a high WRITE rate and high data replication would cause a large amount of overhead traffic. In comparison, low-WRITE-ratio scenarios favors high replication because without worrying about high replication overhead, r's improvement on A is more significant than m's for the same maintenance overhead. Eventually r and m stabilize as discussed before. (4) The throughput performance of low-WRITE-ratio scenarios are the same. As expected, a high-WRITE ratio does degrade the system's throughput performance. A significant drop in the throughput is observed for a load range from 40 to 100 MB/hour. This is because when the system load is relatively small with respect to the system's bandwidth budget, the system can adjust its setting to accommodate the extra WRITE workload. On the other hand, when the load is very high, the system can store and serve only a very small fraction of data objects regardless of the WRITE ratio, so the amount of accepted WRITE workload is negligible. Hence, in these two cases, the WRITE-ratio does not matter. (5) The average sojourn and average storage load in this test are merely the products of the tuning of $\{\eta, \xi, r\}$ and do not contribute

useful information.

4.8.4 Data Size and Throughput Performance

In all the previous tests the system's storage capacity does not constraint the throughput performance. To understand to what degree the data size to be stored will affect the throughput performance, we compare scenarios with $W = \{10M, 1M, 100K, 10K\}$ of $1M$ objects. We plot the optimal system parameters and the resulting DHT performance in Figure 4.8, which reveals the following conclusions: (1) The η curves for scenarios of $W = \{100K, 10K\}$ are identical since in both scenarios the bandwidth budget is the bottleneck resource while the storage budget is relatively abundant. For the scenario of $W = 1M$, the storage constraint kicks in when the system load is smaller than $20MB$/hour so that η always stays above about 0.3; for the same reason, when W increases to $10M$, η always stays above 0.72 for the system load less than $50MB$/hour. (2) For the scenarios of $W = \{1M, 100K, 10K\}$, their ξ curves follow the same trend as the system load increases because in these cases the system's bandwidth constraint is more dominant than the storage constraint. For the $W = 10M$ scenario, the ξ curve drops far below 1.0 even at a very small load before the system's bandwidth budget saturates, and this means the system's storage is overloaded. The figure shows that the storage load at the point when ξ is about to drop below 1.0 is about $5GB$ and the corresponding W is about $1M$ objects $\cdot$ $1MB$/object $= 1TB$. This means, we can use a system storage to store about $W = (10GB/5GB)(1TB) = 2TB$ data when the bandwidth budget is $100MB$ per hour and the per-node storage budget is $10GB$ in this setup. (3) The tuning of r for scenarios of $W \leq 1M$ exhibits the same tendency. For the $W = 10M$ scenario, r is kept lower that others' because reducing the number of replicas can make some storage room to accommodate more unique data objects. Also, the heavy storage load in the scenarios of $W = 1M$ and $10M$ requires the system to use more dynamic nodes. As a result, a higher m is used to counter the resultant node instability since using larger r would produce more data replicas and further stress the system's storage budget. Eventually r and m stabilize because the maintenance overhead associated with a large value of $r \cdot m$ at low node stability is very expensive, and the system can afford this high expense only when the load is small. This test shows that slight storage overload has a negligible impact on the system's throughput performance.

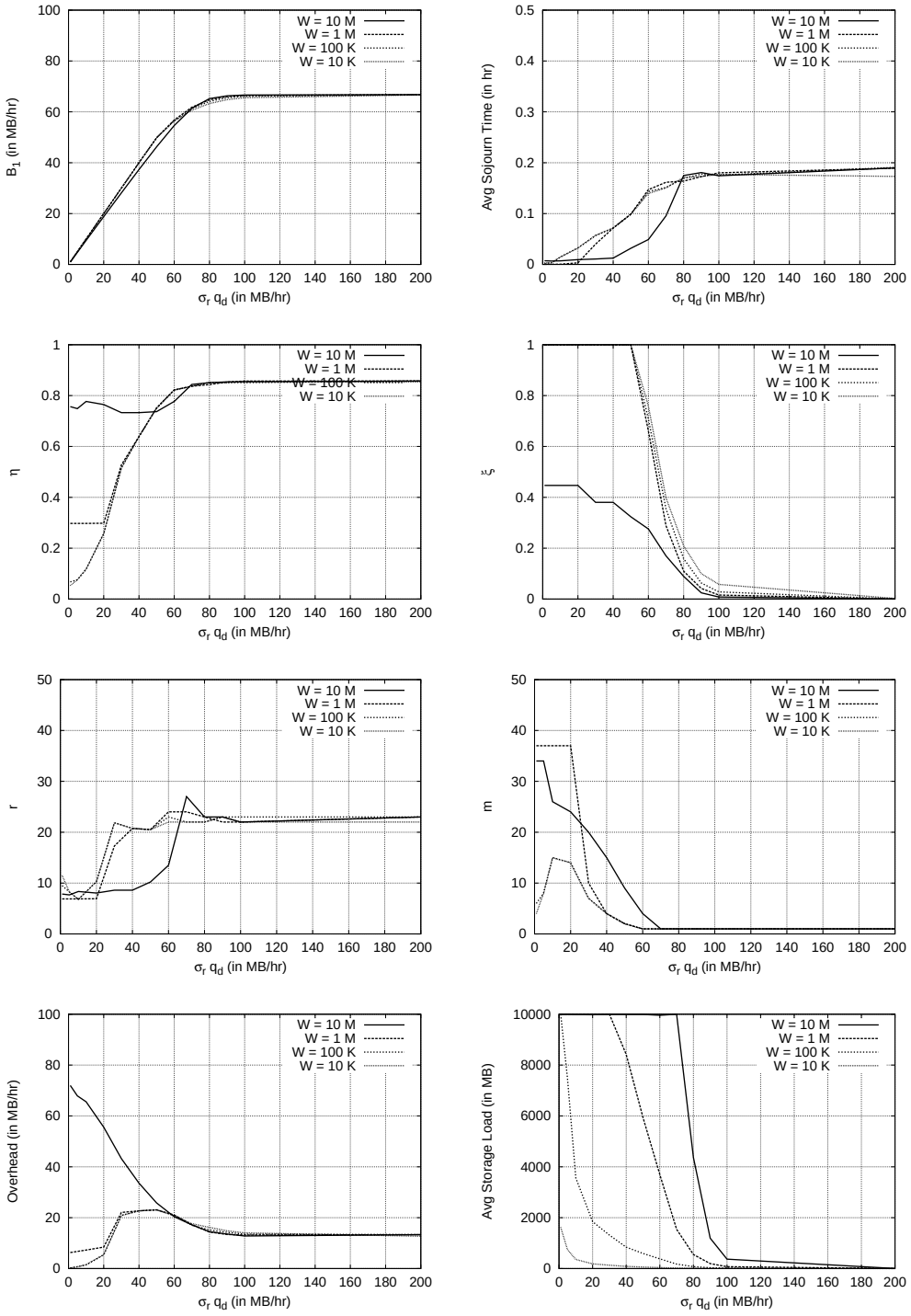

Figure 4.8: Resultant throughput of DHT model for different data size to be stored (W) as we vary the workload on the system.

Chapter 5

Conclusion and Future Work

We have shown in Chapter 3 that in randomized DHTs, routing decisions based on information about direct overlay neighbors do not necessarily lead to shortest overlay routes, and that the performance gap increases in tandem with the churn rate. On the other hand, collecting routing tables of more nodes is not feasible due to the high propagation, storage and forwarding costs. We propose a novel routing protocol, RASTER, which to the best of our knowledge, is the first overlay routing protocol that encodes and aggregates routing information. Exploring the impact of heterogeneity in bitmap resolutions on the system performance will be part of our future work. We also intend to investigate routing based on additional important performance metrics such as end-to-end latency for future work. One possible approach is that instead of recording merely binary values to denote bins' reachability, we can record the minimum latency to each bin. Then, instead of sorting the bitmaps based on their hop-count radius, we sort the bits of each bin based on their hop-count radius scaled with the inverse of the minimum latency to the bin (similar to *delay-aware routing* technique [16]).

We made the case in Chapter 4 that careful design of DHT-based P2P systems can optimize their utility even in the presence of unstable members. Our model highlights that the combination of node selection, content selection, and data redundancy strategies is the major factor in DHT administration. We have observed a few rules of thumb from our model results for the objective of throughput optimization:

- Under dynamic P2P setup, data replication or erasure coding with low fragmentation (for example, $m = 2, 3$) is favored in presence of large user workload.
- Using an appropriate object size is important to the system's throughput performance. An appropriate object size should lead to a relatively insignificant transfer time in comparison to nodes' average session time.
- Do not overload the P2P system since doing so will not lead to a better throughput. We observe that a highly available DHT system does not fully exploit its potential P2P capacity; in fact, not even close.
- The bandwidth constraint is very likely the bottleneck to the throughput objective than the storage constraint. We suggest first improve the system's throughput performance with respect to the bandwidth constraint, then estimate the maximum amount of unique data objects the system could maintain without affecting its throughput performance.

An important observation is that there exists an optimal tuning for the number of most stable nodes, the number of most popular documents and the redundancy parameters to be deployed in a P2P system. The optimal setup, leading to the maximum throughput, the system throughput converges to some value as the load on the system increases. This indicates that there exists a throughput-related limitation for any generic distributed information system in the presence of node instability. Understanding this throughput limitation is important to P2P systems since excellent load scalability is a major motivation that spurts the development and usage of P2P systems. However, to a degree the gain from collective capacity of peer nodes is offset by the cost of maintaining the system in presence of node dynamics. Structured P2P systems may thus not be as scalable as previously perceived.

While we have shown the evolution trends of the system parameters leading to the optimal DHT throughput, verifying these results in realistic P2P setups and applying them to administer DHT systems is a natural extension. Also, extending the current model beyond average case analysis, by studying appropriate distribution of replication parameters among nodes, and accounting for the heterogeneity in node capacity and storage, is another possible extension. One possible future work is to build a model in which nodes leaving the system queue the current requests for later service. Some P2P applications such as P2P collective computing program [7] are able to tolerate a longer response time. In this case,

a P2P node that has undergone its downtime and comes back on-line can still resume the unfinished jobs left in the queue. Another possible future work is to develop a decentralized algorithm to control the system workload and system tuning to match the analysis. Due to the lack of a centralized control infrastructure, the P2P system must need a decentralized control mechanism to measure the system's workload stress over time and then to collaborate nodes to control the flow of incoming workload when the system capacity is stressed.

Bibliography

[1] Bittorrent. http://www.bittorrent.org/.

[2] Gnutella. http://www.gnutella.com.

[3] Kazaa. http://www.kazaa.com.

[4] Napster. http://www.napster.com.

[5] OpenDHT. http://opendht.org/.

[6] Planetlab. http://www.planet-lab.org/.

[7] SETI. http://setiathome.berkeley.edu/.

[8] L. Barriere, P. Fraigniaud, E. Kranakis, and D. Krizanc. Efficient Routing in Networks with Long Range Contacts. In *Proc. of the Fifteenth International Symposium on Distributed Computing (DISC 2001)*, Lisboa, Portugal, 2001.

[9] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *Proc. of IPTPS*, Berkeley, CA, USA, 2003.

[10] C. Blake and R. Rodrigues. High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two. In *Proc. of HotOS*, Lihue, Hawaii, USA, 2003.

[11] M. Castro, P. Drschel, Y. C. Hu, and A. I. T. Rowstron. Topology-Aware Routing in Structured Peer-to-Peer Overlay Networks. In *Proc. of Intl. Workshop on Future Directions in Distrib. Computing*, Bertinoro, Italy, 2002.

[12] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. High-bandwidth Content Distribution in a Cooperative Environment. In *Proc. of IPTPS*, Berkeley, CA, USA, 2003.

[13] Byung-Gon Chun, B. Y. Zhao, and J. D. Kubiatowicz. Impact of Neighbor Selection on Performance and Resilience of Structured P2P Networks. In *Proc. of IPTPS*, Ithaca, NY, USA, 2005.

[14] D. Coppersmith, D. Gamarnik, and M. Sviridenko. In *Proc. of ACM/SIAM SODA), Title = "The Diameter of a Long-range Percolation graph", Year=2002*, San Francisco, CA, USA.

[15] F. Dabek, F. Kaashoek, D. Karger, R. Morris, and I.Stoica. Wide-Area Cooperative Storage with CFS. In *Proc. of ACM SOSP*, Chateau, Banff, Canada, 2001.

[16] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris. Designing a DHT for Low Latency and High Throughput. In *Proc. of NSDI*, Berkeley, CA, USA, 2003.

[17] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a Common API for Structured Peer-to-peer Overlays. In *Proc. of IPTPS*, Berkeley, CA, USA, 2003.

[18] P. Fraigniaud and P. Gauron. The Content Addressable D2B. In *Technical Report, CNRS Universie de Paris Sud*, Paris, France, 2003.

[19] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I Stoica. Load Balancing in Dynamic Structured P2P Systems. In *Proc. of IEEE INFOCOM*, Hong Kong, China, 2004.

[20] P. Godfrey and I. Stoica. Heterogeneity and Load Balance in Distributed Hash Tables. In *Proc. of IEEE INFOCOM*, Miami, Fl, USA, 2005.

[21] P. B. Godfrey, S. Shenker, and I. Stoica. Minimizing Churn in Distributed Systems. In *Proc. of ACM SIGCOMM*, Pisa, Italy, 2006.

[22] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *Proc. of ACM SOSP*, New York, NY, USA, 2003.

[23] R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proc. of ACM SIGCOMM*, Karlsruhe, Germany, 2003.

[24] N. Harvey, M. Jones, S. Saroiu, M. theimer, and A. Wolman. Skipnet: A Scalable Overlay Network with Practical Locality Properties. In *Proc. of 4th USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, USA, 2003.

[25] F. Kaashoek and D. karger. Koorde: A Simple Degree-optimal Hash Table. In *Proc. of IPTPS*, Berkeley, CA, USA, 2003.

[26] D. R. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proc. of ACM STOC*, El Paso, TX, USA, 1997.

[27] L. Kelinrock. *Queueing Systems*, volume 2. John Wiley and Sons, 1976.

[28] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proc. of ACM STOC*, Portland, Oregon, USA, 2000.

[29] A. Klemm, C. Lindemann, M. K. Vernon, and O. P. Waldhorst. Characterizing the Query Behavior in Peer-to-Peer File Sharing Systems. In *Proc. of IMC*, Varna, Bulgaria, 2004.

[30] S. Krishnamurthy, S El-Ansary, E. Aurell, and S. Haridi. A Statistical Theory of Chord under Churn. In *Proc. of IPTPS*, Ithaca, NY, USA, 2005.

[31] D. Leonard, V. Rai, and D. Loguinov. On Lifetime-based Node Failure and Stochastic Resilience of Decentralized Peer-to-peer Networks. In *Proc. of SIGMETRICS*, Banff, Alberta, Canada, 2005.

[32] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience. In *Proc. of ACM SIGCOMM*, Karlsruhe, Germany, 2003.

[33] D. Malkhi, , M. Naor, and D. Ratajczak. Viceroy: A Scalable Dynamic Emulation of the Butterfly. In *Proc. of ACM PODC*, Monterey, CA, USA, 2002.

[34] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small World. In *Proc. of 4th USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, USA, 2003.

[35] G. S. Manku, M. Bawa, and P. Raghavan. Know Thy Neighbor's Neighbor: the Power of Lookahead in Randomized P2P Networks. In *Proc. of ACM STOC*, Chicago, IL, USA, 2004.

[36] J. Mickens and B. Noble. Predicting Node Availability in Peer-to-Peer Networks. In *Proc. of SIGMETRICS POSTER*, Banff, Alberta, Canada, 2005.

[37] S. Milgram. The Small World Problem. In *Psychology Today*, 1967.

[38] M. Naor and U. Wieder. Novel Architectures for P2p Applications: the Continuous-Discrete Approach. In *Proc. of ACM SPAA*, San Diego, CA, USA, 2003.

[39] National Institute of Standards and Technology. FIPS 180-1. Secure hash Standard. http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[40] D. Nowell, H. Balakrishnan, and D. Karger. Analysis on the Evolution of Peer-to-peer Systems. In *Proc. of ACM PODC*, Cambridge, MA, USA, 1992.

[41] D. Nowell, H. Balakrishnan, and D. Karger. Observations on the Dynamic Evolution of Peer-to-Peer Networks. In *Proc. of IPTPS*, Cambridge, MA, USA, 2001.

[42] C. Plaxton, R. Rajaraman, and A. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Proc. of ACM SPAA*, Newport, RI, USA, 1997.

[43] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. of ACM SIGCOMM*, San Diego, CA, USA, 2001.

[44] S Rhea, B. Chun, J. Kubiatowicz, and S. Shenker. Fixing the Embarrassing Slowness of OpenDHT on PlanetLab. In *Proc. of USENIX WORLDS*, San Francisco, CA, USA, 2005.

[45] S. Rhea, B. Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. OpenDHT: A Public DHT Service and Its Uses. In *Proc. of ACM SIGCOMM*, Philadelphia, PA, USA, 2005.

[46] J. Roberts, U. Mocci, and J. Virtamo. *Broadband Network Teletraffic, Final Report of Action COST 242*. Springer, 1996.

[47] R. Rodrigues and C. Blake. When Multi-hop Peer-to-peer Lookup Matters. In *Proc. of IPTPS*, La Jolla, CA, USA, 2004.

[48] R. Rodrigues and B. Liskov. High Availability in DHTs: Erasure Coding vs. Replication. In *Proc. of IPTPS*, Ithaca, NY, USA, 2005.

[49] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, 2001.

[50] S. Saroiu, P. Gummadi, and S. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proc. of ACM MMCN*, San Jose, Ca, USA, 2002.

[51] I. Stoica, R. Morris, David Liben-Nowell, D. Karger, F. Kaashoek, and H. Balarkrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *IEEE Transactions on Networking*, 11, 2003.

[52] J. Travers and S. Milgram. An Experimental Study of the Small World Problem. In *Sociometry*, 1969.

[53] D. Watts and S. Strogatz. Collective Dynamics of Small-World Networks. In *Nature*, 1998.

[54] H. Weatherspoon, B. G. Chun, C. W. So, and J. Wubiatowicz. Long-term Data Maintenance: A Quantitative Approach. In *Technical Report UCB/CSD-05-1404*, Berkeley, CA, USA, 2005.

[55] H. Zhang, A. Goel, and R. Govindan. Incrementally Improving Lookup Latency in Distributed Hash Table Systems. In *Proc. of ACM SIGMETRICS*, San Diego, CA, USA, 2003.

[56] B. Y. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22, 2004.

[57] Shelley Q. Zhuang, Dennis Geels, Ion Stoica, and Randy H. Katz. On Failure Detection Algorithms in Overlay Networks. In *Proc. of IEEE INFOCOM*, Miami, Florida, USA, 2005.