A performance vs. cost framework for evaluating DHT design tradeoffs under churn

Jinyang Li, Jeremy Stribling, Robert Morris, M. Frans Kaashoek and Thomer M. Gil
{jinyang, strib, rtm, kaashoek, thomer}@csail.mit.edu
MIT Computer Science and Artificial Intelligence Laboratory

***Abstract*— Protocols for distributed hash tables (DHTs) incorporate features to achieve low latency for lookup requests in the face of *churn*, continuous changes in membership. These protocol features can include a directed identifier space, parallel lookups, pro-active flooding of membership changes, and stabilization protocols for maintaining accurate routing. In addition, DHT protocols have parameters that can be tuned to achieve different tradeoffs between lookup latency and communication cost due to maintenance traffic. The relative importance of the features and parameters is not well understood, because most previous work evaluates protocols on static networks.**

This paper presents a performance versus cost framework (PVC) that allows designers to compare the effects of different protocol features and parameter values. PVC views a protocol as consuming a certain amount of network bandwidth in order to achieve a certain lookup latency, and helps reveal the *efficiency* with which protocols use additional network resources to improve latency. To demonstrate the value of PVC, this paper simulates Chord, Kademlia, Kelips, OneHop, and Tapestry under different workloads and uses PVC to understand which features are more important under churn. PVC analysis shows that the key to efficiently using additional bandwidth is for a protocol to adjust its routing table size. It also shows that routing table stabilization is wasteful and can be replaced with opportunistic learning through normal lookup traffic. These insights combined demonstrate that PVC is a valuable tool for DHT designers.

I. Introduction

Designing a DHT lookup protocol for *static* networks is well-understood. Typical designs forward lookups for keys through routing tables that point to other hosts. Different DHTs form different topologies with these routing tables, which result in different asymptotic tradeoffs of the amount of per-node state and the number of network hops during lookups. Most DHTs find low latency neighbors using techniques such as Proximity Neighbor Selection (PNS) [1]. With a few exceptions [2]–[5], most previous work evaluates lookup latency in a static network, as described in Section VI on related work.

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

One of the main contributions of this paper is a simple performance versus cost framework (PVC).[1] PVC measures cost as the amount of network traffic that a DHT's lookups and maintenance traffic incur. For its performance metric, PVC measures the failure rate and latency of DHT lookups. PVC helps designers compare the effects of different design choices in each DHT through the systematic exploration of various combinations of protocol parameters. It is often misleading to evaluate the performance benefits of any one design choice in isolation, as all design choices improve performance at the cost of extra communications. To design better DHTs, we need to understand which design choice is *more* efficient than others at using extra communication bits. For example, the methodology allows a comparison of fast stabilization and parallel lookups in a way that considers both their ability to reduce latency and the fact that both techniques increase bandwidth consumption.

Another contribution of this paper is an extensive PVC-based simulation study of a wide range of DHT protocols (Chord [7], Kademlia [8], Kelips [9], OneHop [10], and Tapestry [11]). The study explores how the protocols' design choices and parameters affect their efficiency under churn. Table I summarizes the paper's observations about DHT design choices and parameters. This case study of existing DHTs using PVC reveals that the key to efficient bandwidth use is for a DHT node to adjust its routing table size in response to extra bandwidth and churn. PVC also shows that opportunistic learning through existing DHT lookup traffic is more efficient than active routing table stabilization. These results taken together demonstrate the value of PVC as a tool to design and evaluate DHT protocols.

This study does not model storage or transmission of DHT data. Under high churn, the cost of keeping DHT data available might dwarf the routing table maintenance cost. A different set of techniques such as replication with lazy repair [12] can be used to reduce DHT data maintenance cost under churn. Additionally, the simulator used in this study does not model queuing delay or packet loss, and thus cannot evaluate protocol features intended to reduce congestion.

The rest of the paper is organized as follows. Section II motivates and presents PVC. Section III summarizes the salient properties of the DHTs this paper compares, and Section IV describes the experimental methodology. Section V evaluates

[1] We introduced a preliminary version of PVC in a position paper [6].

Section	Insights
V	Minimizing lookup latency requires complex workload-dependent parameter tuning.
V-A	The ability of a protocol to control its bandwidth usage has a direct impact on its scalability and performance under different network sizes.
V-B	DHTs that distinguish between state used for the correctness of lookups and state used for lookup performance can more efficiently achieve low lookup failure rates under churn.
V-C	The strictness of a DHT protocol's routing distance metric, while useful for ensuring progress during lookups, limits the number of possible paths, causing poor performance under pathological network conditions such as non-transitive connectivity.
V-D	Increasing routing table size to reduce the number of expected lookup hops is a more cost-efficient way to cope with churn-related timeouts than stabilizing more often.
V-E	Issuing copies of a lookup along many paths in parallel is more effective at reducing lookup latency due to timeouts than faster stabilization under a churn intensive workload.
V-F	Learning about new nodes during the lookup process can essentially eliminate the need for stabilization in some workloads.
V-G	Increasing the rate of lookups in the workload, relative to the rate of churn, favors all design choices that reduce the overall lookup traffic. For example, one should use extra state to reduce lookup hops (and hence forwarded lookup traffic). Less lookup parallelism is also preferred as it generates less redundant lookup traffic.

TABLE I

INSIGHTS OBTAINED BY USING PVC TO EVALUATE A SET OF PROTOCOLS (CHORD, KADEMLIA, KELIPS, ONEHOP AND TAPESTRY).

the impact of different design decisions on performance under churn. Section VI relates this paper's study to previous studies. Finally, Section VII summarizes our findings.

II. PVC: A PERFORMANCE VS. COST FRAMEWORK

The goal of PVC is to address two challenges in evaluating lookup protocols for DHTs. First, most protocols can be tuned to have low lookup latency by including features such as aggressive membership maintenance, faster routing state refreshes, parallel lookups, or a more thorough exploration of the network to find low latency neighbors. Not only do these features make straightforward comparisons of the performance of different DHT protocols difficult, but they also complicate the evaluation of new features, since protocol features typically improve lookup performance by consuming resources. Thus a comparison of DHT protocols must evaluate the *efficiency* with which they exploit additional resources to reduce latency. A good framework should use a metric which quantifies the *cost*, as well as the performance, of protocols.

The second challenge is coping with each protocol's set of tunable parameters (*e.g.*, stabilization interval, etc.). The best parameter values for a given workload are often hard to predict, so there is a danger that a performance evaluation might reflect the evaluator's parameter choices more than it reflects the underlying algorithm. In addition, parameters often correspond to a given protocol feature. A good framework should allow designers to judge the extent to which each parameter (and thus each feature) contributes to an efficient performance vs. cost tradeoff.

A. The PVC Approach

In response to these two challenges, we propose PVC, a performance vs. cost framework and evaluation methodology for assessing DHT protocols, comparing different design choices and evaluating new features.

PVC uses the average number of bytes sent per node per unit time as the cost metric. This cost accounts for all messages sent by a node, including periodic routing table refresh traffic, lookup traffic, and join traffic. PVC ignores state storage costs (*e.g.*, the size of each node's routing table) because communication is typically far more expensive than storage. The main cost of state is often the communication cost necessary for maintaining the correctness of that state.

In PVC, nodes try to forward lookups to the node responsible for the lookup key. The identity of the responsible node is returned to the sender as the result of the lookup. A lookup is considered failed if it returns the wrong node among the current set of participating nodes (i.e. those that have completed the join procedure correctly) at the time the sender receives the lookup reply, or if the sender receives no reply within some timeout window. PVC uses two metrics to characterize a DHT's performance: median latency of successful lookups and lookup failure rate. PVC only incorporates lookup hop-count indirectly, to the extent that it contributes to latency. In the presence of churn, routing tables tend to become incorrect or out of date, causing lookups to suffer timeouts or completely fail. DHTs often recover from lookup timeouts by retrying the lookup through an alternate neighbor.

In the interest of a fair comparison, DHTs should follow a few common guidelines for dealing with lookup timeouts. In PVC experiments, all protocols time out individual messages after an interval of three times the round-trip time to the target node, though more sophisticated techniques are possible [3], [4], [13], [14]. Following the conclusions of previous studies [3], [4], a node encountering a timeout to a particular neighbor during a lookup does not immediately declare that neighbor dead; the lookup proceeds to an alternate node if one exists, and recovery does not begin for the failed neighbor until several RPC timeouts to that neighbor occur.

All protocols retry lookups for up to a maximum of four seconds, after which PVC declares that the lookup has failed. This definition of failure is arbitrary: a shorter maximum time would decrease average latency while increasing failure rate, while a longer maximum would increase average latency while decreasing the failure rate. Further, each failed lookup contributes a disproportionate four seconds to the average lookup latency statistic. For these reasons PVC measures lookup failure rate and *median* lookup latency as separate

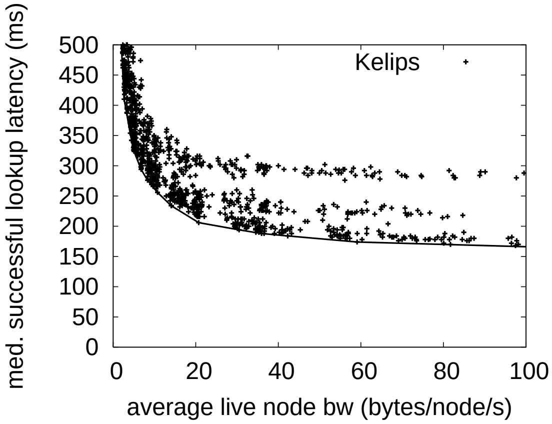


Fig. 1. Performance vs. cost tradeoff in Kelips, churn intensive workload. Each point represents the median lookup latency of successful lookups vs. the communication cost achieved for a unique set of parameter values. The convex hull (solid line) represents the best achievable performance/cost combinations.

performance metrics.

PVC needs a workload. The amount a protocol must communicate to keep node routing tables up to date depends on how frequently nodes join and crash (the churn rate). For the most part, the total bandwidth consumed by a protocol is a balance between table maintenance traffic and lookup traffic, so the main characteristic of a workload is the relationship between lookup rate and churn rate. This paper investigates two workloads, one that is churn intensive and one that is lookup intensive.

B. Overall Convex Hulls

To find the best performance/cost tradeoff for a DHT with many tunable parameters, PVC systematically simulates the DHT with different combinations of parameter values. PVC measures the performance and cost for each combination; this paper shows the results as graphs with average bandwidth usage on the x-axis and median lookup latency or failure rate on the y-axis. For example, Figure 1 shows bandwidth and latency for Kelips with different parameter combinations under a particular workload. There is no single best parameter combination for this workload. Instead, there is a set of most efficient combinations: for each cost, there is a smallest achievable lookup latency, and for each lookup latency, there is a smallest achievable communication cost. The curve connecting these most efficient points is the overall *convex hull* segment (shown by the solid line in Figure 1). PVC uses convex hulls to find the best performance/cost tradeoffs for a given workload. It is possible that better combinations exist but that PVC failed to find them.

The convex hull in Figure 1 outlines the most efficient parameter combinations found by PVC. A DHT operator would have to adjust these parameters manually in the absence of a self-tuning protocol [4], [15].

C. Parameter Convex Hulls

Figure 1 shows the combined effect of many parameters. PVC can also be used to evaluate whether a particular parameter is more important to tune than others in order to achieve the best performance/cost tradeoff. This is done by calculating a set of convex hulls, one for each value of the parameter under study. Each convex hull is generated by fixing the parameter of interest and varying all others. The positions of these parameter convex hulls relative to the overall convex hull indicates the relative "importance" of the parameter: the performance benefit obtainable by tuning the parameter to consume more bandwidth.

Figure 2 presents parameter convex hulls for two different parameters, each compared with the overall convex hull. The parameter in Figure 2(a) has a single best value (32) for this workload. The parameter in Figure 2(b) is comparatively much more important to tune. No single parameter convex hull lies entirely along the overall hull; rather, the overall hull is made up of segments from different parameter hulls. This suggests the parameter should be tuned based on the application's desired latency/bandwidth tradeoff.

One can quantify the importance of a particular parameter value by calculating the area between the parameter's hulls and the overall convex hull over a fixed range of the x-axis (the cost range of interest). Figure 3 shows an example. The smaller the area, the more closely a parameter hull approximates the best overall hull. The minimum area over all of a parameter's values indicates how important it is to tune the parameter. The bigger the minimum area, the more important the parameter since there is a larger potential for inefficiency by setting the parameter to a single value. Figure 2(a) shows a parameter with nearly zero minimum area, while Figure 2(b) shows a parameter with a large minimum area. Hence, it is relatively more important to tune the latter.

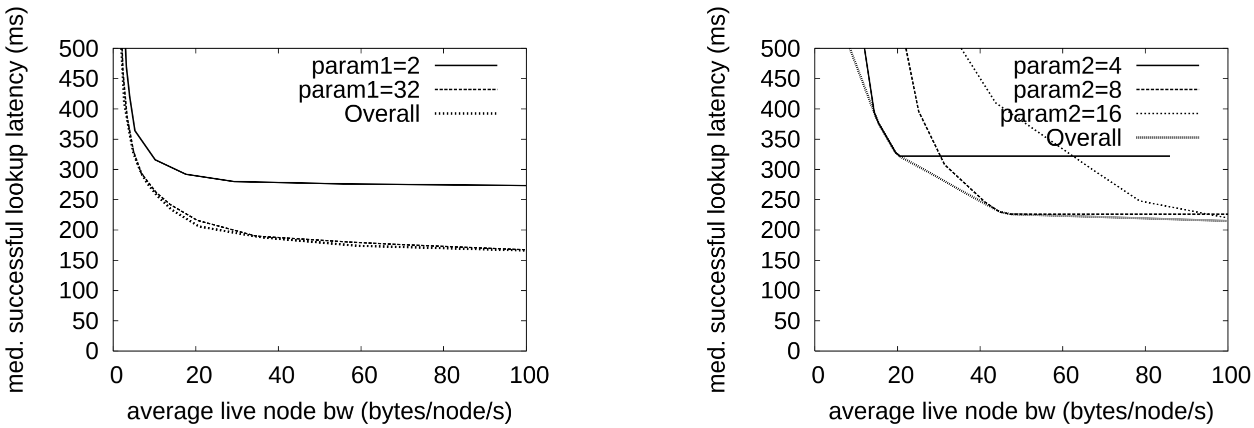
There is a relationship between this notion of parameter importance and the efficiency of DHT mechanisms. Suppose that an important parameter affects how much network bandwidth a particular DHT mechanism consumes; for example, the parameter might control how often a DHT stabilizes its routing table entries. If more network capacity becomes available, then any re-tuning of the DHT's parameters to make best use of the new capacity will likely require tuning this important parameter. That is, important parameters have the most effect on the DHT's ability to use *extra* communication bandwidth to achieve low latency, and in that sense important parameters correspond to efficient DHT mechanisms.

III. Protocol Overviews

This paper studies design choices made by five existing protocols: Tapestry [11], Chord [7], Kelips [9], Kademlia [8] and OneHop [10]. This section provides brief overviews of each DHT, identifying protocol parameters that relate to the design choices under study.

A. Tapestry

The ID space in Tapestry is structured as a tree. A Tapestry node ID can be viewed as a sequence of l base-b digits. The node the maximum number of matching prefix digits with the key is the node responsible for the key. In a network of n nodes, each node's routing table contains approximately



(a) (b)

Fig. 2. Convex hull segments for two different parameters, in comparison with the overall convex hull. The left graph shows a parameter that has a single fixed value that achieves best performance. The right graph shows a parameter that must take on different values to achieve different best performance/cost tradeoffs.

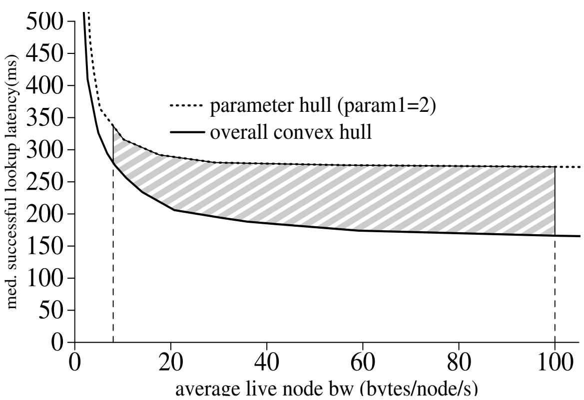


Fig. 3. The efficiency of param1 with a value of 2 (from Figure 2(a)) compared with the efficiency of the overall convex hull. The striped area between the two curves represents the efficiency difference between param1=2 and the best configurations between a cost range of 8 to 100 bytes/node/s.

Parameter		**Range**
Base	b	2 – 128
Stabilization interval	t_{stab}	18 sec – 19 min
Number of backup nodes	n_{redun}	1 – 8
Number of nodes contacted during repair	n_{repair}	1 – 10

TABLE II

TAPESTRY PARAMETERS

$\log_b(n)$ levels, each with b distinct ID prefixes. Nodes in the m^{th} level share a prefix of length $m-1$ digits, but differ in the m^{th} digit. Each entry may contain up to n_{redun} nodes, sorted by latency. The closest of these nodes is the entry's *primary neighbor*; the others serve as *backup neighbors*. Tapestry uses a nearest neighbor algorithm [16] to populate its routing table entries with nearby nodes.

Nodes forward a lookup by resolving successive digits in the lookup key (*prefix-based routing*). When no more digits can be resolved, an algorithm known as *surrogate routing* determines exactly which node is responsible for the key [11]. Routing in Tapestry is recursive.

For lookups to be correct, at least one neighbor in each routing prefix must be alive. Tapestry checks the liveness of each primary neighbor every t_{stab} seconds. If the node is found to be dead, the next closest backup in that entry (if one exists) becomes the primary. When a node declares a primary neighbor dead, it contacts some number of other neighbors (n_{repair}) asking for a replacement. Table II lists Tapestry's parameters varied in our simulations.

B. Chord

Chord structures its identifiers in a circle. The node responsible for a key y is its successor (*i.e.*, the first node whose ID is equal to k, or follows y in the ID space) using consistent hashing [17]. In Chord, a lookup for a key terminates at the key's *predecessor*, the node whose ID most closely precedes the key and it returns the successor as the lookup's value. A node in base-b Chord keeps $(b-1)\log_b(n)$ fingers whose IDs lie at exponentially increasing fractions of the ID space away from itself. Each node keeps a successor list of n_{succ} nodes. Chord uses the Proximity Neighbor Selection (PNS) method discussed in [1], [13], [18]. To obtain each i^{th} PNS finger, a node retrieves the successor list of n_{succ} nodes from a node with ID $(\frac{b-1}{b})^{i+1}$ away from itself and chooses the node closest in network latency to itself as the i^{th} PNS finger. Chord can route either iteratively or recursively [7]; this paper presents results for the latter.

A Chord node checks all its fingers for liveness every t_{finger} seconds. For each finger found dead, the node issues a lookup for a replacement PNS finger. A node separately stabilizes its successor list periodically (t_{succ}) by retrieving and merging its successors' predecessor and successor lists. Table III lists the Chord parameters that we vary in our simulations.

Parameter		Range
Base	b	2 – 128
Finger stabilization interval	t_{finger}	18 sec – 19 min
Number of successors	n_{succ}	8,16,32
Successor stabilization interval	t_{succ}	18 sec – 4.8 min

TABLE III

CHORD PARAMETERS

Parameter		Range
Gossip interval	t_{gossip}	10 sec – 19 min
Group ration	r_{group}	8, 16, 32
Contact ration	$r_{contact}$	8, 16, 32
Contacts per group	$n_{contact}$	2, 8, 16, 32
Routing entry timeout	t_{out}	6, 18, 30 min

TABLE IV

KELIPS PARAMETERS

Parameter		Range
Nodes per entry	k	2 – 32
Parallel lookups	α	1 – 32
Number of IDs returned	n_{tell}	2 – 32
Stabilization interval	t_{stab}	4 – 19 min

TABLE V

KADEMLIA PARAMETERS

Parameter		Range
Slices	n_{slices}	3,5,8
Units	n_{units}	3,5,8
Ping/Aggregation interval	t_{stab}	4 sec – 64 sec

TABLE VI

ONEHOP PARAMETERS

C. Kelips

Kelips divides the identifier space into $g \approx \sqrt{n}$ groups. A node's group is its ID modulo g. Each node's routing table contains an entry for each other node in its own group, and $n_{contact}$ "contact" nodes from each of the foreign groups. Thus a node's routing table size is $\sqrt{n} + n_{contact} * (\sqrt{n} - 1)$ nodes in a network of n nodes.

Kelips does not define an explicit mapping of a given key to its responsible node. Kelips replicates key/value pairs among all nodes within a key's group and a lookup terminates whenever it reaches a node storing the corresponding key/value pair. Lookups for non-existent keys have higher latency than lookups that do have values stored under the keys as lookups cannot terminate efficiently by reaching their responsible nodes. For this reason, the variant of Kelips in this paper defines lookups only for IDs of node that are currently in the network. The originating node executes a lookup for a key by asking a contact in the key's group for target key's node ID, and then (iteratively) contacting that node. If that fails, the originator tries routing the lookup through other contacts for that group, and then through randomly chosen routing table entries. In a static network, a Kelips lookup should take two hops.

Nodes gossip periodically every t_{gossip} seconds. A node chooses one random contact and one node within the same group to send a random list of r_{group} nodes from its own group and $r_{contact}$ contact nodes. Routing table entries that have not been refreshed for t_{out} seconds expire. Nodes learn round trip times (RTTs) and liveness information from each RPC, and preferentially route lookups through low RTT contacts. Table IV lists the parameters we use for Kelips. We use $g = \sqrt{1000} \approx 32$ in our Kelips simulations with $n = 1000$ nodes.

D. Kademlia

Kademlia structures its ID space as a tree. The distance between two keys in ID space is their exclusive or, interpreted as an integer. The k nodes whose IDs are closest to a key y store a replica of y. A node's routing table keeps $\log_2(n)$ buckets that each stores up to k node IDs sharing the same binary prefix of a certain length.

Kademlia performs iterative lookups: a node x starts a lookup for key y by sending parallel lookup RPCs to the α nodes in x's routing table whose IDs are closest to y. A node replies to a lookup RPC by sending back a list of the n_{tell} nodes it believes are closest to y in ID space. Node x always tries to keep α outstanding RPCs. A lookup terminates when some node replies with key y, or until the last k nodes whose IDs are closest to y did not return any new node ID closer to y. Like Kelips, Kademlia also does not have an explicit mapping of a key to its responsible node, therefore terminating lookups for non-existent keys requires extra communication with the last k nodes. For this reason, we also use node IDs as lookup keys in Kademlia experiments and the last step in a lookup is an RPC to the target node. Our Kademlia implementation favors proximate nodes. With each lookup RPC, a node learns RTT information for existing routing neighbors or previously unknown nodes to be stored in its routing bucket. A node periodically (t_{stab}) examines all of its routing buckets and performs a lookup for each bucket's binary prefix if there has not been a lookup through it since the last stabilization. Kademlia's stabilization only ensures that at least *one* entry in each bucket was alive in the past t_{stab} seconds, while stabilization in Tapestry (Chord) ensures *all* routing entries were alive in the past t_{stab} (t_{finger}) seconds. Table V summarizes the parameters varied in our Kademlia simulations.

E. OneHop

In OneHop, a node knows about every other node in the network in order to maintain a lookup hop-count of one, therefore there is no parameter determining how much state a node keeps. Similar to Chord, OneHop [10] assigns a key to its successor node on the ID circle using consistent hashing [17]. The ID space is divided into n_{slice} slices and each slice is further divided into n_{unit} units. Each unit and slice has a corresponding leader. OneHop pro-actively disseminates information regarding all join and crash events to all nodes in the system through the hierarchy of slice leaders and unit leaders. A node periodically (t_{stab}) pings its successor and predecessor and notifies its slice leader of the death of successor/predecessor. A newly joined node sends a live notification event to its slice leader. A slice leader aggregates notifications within its slice and periodically (t_{stab}) informs all other slice leaders about notifications since the last update. A slice leader disseminates notifications from within its slice and from other slices to each unit leader in its own slice. Notifications are further propagated to all nodes within a unit through piggy-backing on each node's ping messages. Table VI summarizes the OneHop parameters varied.

Table VII summarizes the correspondence between design choices and parameters for all the protocols used in this paper.

IV. Experimental Methodology

We implemented the five DHTs in a discrete-event packet level simulator, p2psim. The simulated network, unless otherwise noted, consists of 1024 nodes with a pairwise latency matrix derived from measuring the inter-node latencies of 1024 DNS servers using the King method [19]. The median round-trip delay between node pairs is 156 ms and the average is 178 ms. Since each lookup for a random key must terminate at a specific, random node in the network, the median latency of the topology serves as a lower bound for the median DHT lookup latency. The simulator does not simulate link transmission rate or queuing delays, because the experiments involve only key lookups as opposed to data retrieval.

Each node alternately crashes and re-joins the network; the interval between successive events for each node is exponentially distributed with a mean of one hour. The choice of mean session time is consistent with past studies of peer-to-peer networks [20]. Each time a node joins, it uses a different IP address and DHT identifier. We experimented with two types of workloads: *churn intensive* and *lookup intensive*. In the churn intensive workload, each node issues lookups for random keys at intervals exponentially distributed with a mean of 600 seconds. In the lookup intensive workload, the lookup interval mean is 9 seconds. Unless otherwise noted, all figures are for simulations done in the churn intensive workload. Each simulation runs for six hours of simulated time; statistics are collected only during the second half of the simulation.

For all the graphs below, the x-axis shows the total number of bytes sent by all nodes divided by the sum of the amounts of time that the nodes were up. That is, the x-axis shows the average bytes per second sent by live nodes. This includes all messages sent by nodes, such as lookup, join and routing table maintenance traffic. The size in bytes of a message is counted as 20 bytes (for packet overhead) plus 4 bytes for each IP address or node identifier mentioned in the message. The y-axis indicates lookup performance either in median lookup latency or failure rate.

V. Results

We ran simulations of each protocol with all combinations of the parameters enumerated in Section III. Figures 4 and 5 present the convex hulls of all five DHTs for failure rate and latency, respectively. The convex hulls' overall characteristics are similar in the sense that both failure rate and lookup latency decrease as the protocols consume more bandwidth. However, at any fixed bandwidth cost, the best achievable failure rates

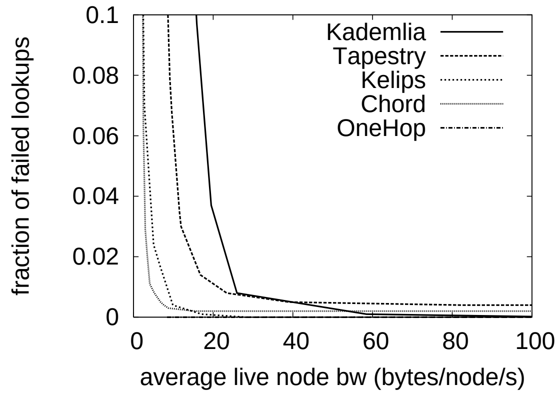


Fig. 4. Overall convex hulls for lookup failure rate for all DHTs, under the churn intensive workload. The failure rate for OneHop is nearly zero for all points on its hull.

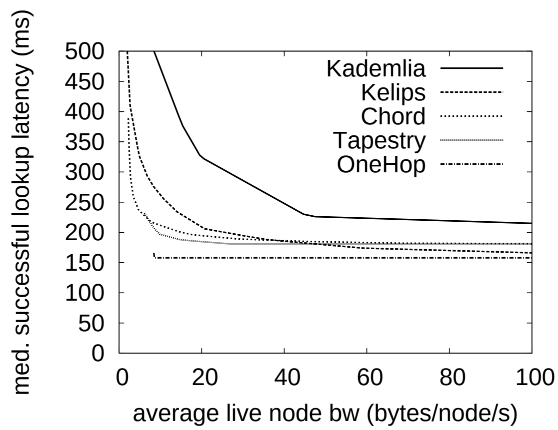


Fig. 5. Overall convex hulls for median successful lookup latency for all DHTs, under the churn intensive workload.

or latencies of different DHTs can differ significantly. For example, at 10 bytes/node/s, OneHop achieves 160ms median lookup latency and Kademlia achieves 450ms with the best parameter settings.

The convex hulls in Figures 4 and 5 are essentially the result of an exhaustive search for the best parameter values. Even on a single protocol's convex hull, the parameter values that provide the most efficient latency/bandwidth or failure rate/bandwidth tradeoff are different in different regions.

In a real deployment, the protocol designer or deployer would have to tune the parameters manually to find the best values, or settle for default values. Tables VIII and IX summarize the importance of tuning each parameter of each protocol with respect to failure rate and lookup latency, respectively. The table lines are ranked most important to least important, as judged by the area between the overall convex hull and the parameter convex hull for the best parameter value (see Section II-C). The area between hulls measures the relative importance of parameters on *average* over a *range* of costs ($1 - 100$ bytes/node/s). To make the effects of tuning a particular parameter more concrete, we also examine the actual performance range achieved by different parameter settings at a *specific* bandwidth cost. In Tables VIII and IX, the **best** and

Design Choices	**Tapestry**	**Chord**	**Kademlia**	**Kelips**	**OneHop**
Separation of lookup correctness from performance	-	t_{succ}	-	-	-
Amount of state	b, n_{redun}	b	k	$n_{contact}$	-
Freshness of state	t_{stab}	t_{finger}	t_{stab}	t_{gossip}	t_{stab}
Lookup parallelism	-	-	α, n_{tell}	-	-
Learning new nodes from lookups	-	-	yes	-	yes

TABLE VII

DESIGN CHOICES AND THEIR CORRESPONDING PARAMETERS.

	Tapestry			**Chord**			**Kelips**			**Kademlia**		
	param	**best**	**worst**	**param**	**best**	**worst**	**param**	**best**	**worst**	**param**	**best**	**worst**
1	t_{stab}	18s:0.005	1152s:0.079	t_{succ}	18s:0.002	288s:0.062	t_{gossip}	10s:0.000	1152s:0.435	n_{tell}	4:0.006	2:0.239
2	n_{redun}	8:0.005	1:0.037	t_{finger}	144s:0.002	1152s:0.006	$n_{contact}$	8:0.000	32:0.006	k	2:0.006	32:0.052
3	n_{repair}	3:0.005	1:0.006	b	16:0.002	8:0.003	t_{out}	360s:0.000	1800s:0.000	α	4:0.006	1:0.042
4	b	2:0.005	128:0.096	n_{succ}	8:0.002	32:0.003	$r_{contact}$	2:0.000	64:0.001	t_{stab}	1152s:0.006	288s:0.199
5							r_{group}	2:0.000	64:0.001			

TABLE VIII

THE RELATIVE IMPORTANCE OF EACH PARAMETER ON A DHT'S FAILURE RATE VS. COST TRADEOFF.
Each protocol's parameters are ranked most important to least important, as determined by the minimum area difference between any of a parameter's values and the overall convex hull (see Figure 3), for costs ranging between 1 and 100 bytes/node/s. To give intuition as to how each parameter affects the failure rate, we pick an example bandwidth value (40 bytes/node/s) and in the **best** column show the value for that parameter resulting in the lowest failure rate, followed by the lowest failure rate achieved at that bandwidth. Similarly, the **worst** column shows the value for that parameter resulting in the highest failure rate, followed by the *best* failure rate achieved with that value while tuning *other* parameters freely.

	Tapestry			**Chord**			**Kelips**			**Kademlia**		
	param	**best**	**worst**	**param**	**best**	**worst**	**param**	**best**	**worst**	**param**	**best**	**worst**
1	b	32:181	128:199	b	32:186	2:226	t_{gossip}	18s:185	1152s:667	n_{tell}	8:247	32:519
2	t_{stab}	144s:181	1152s:195	t_{finger}	72s:186	1152s:218	$r_{contact}$	64:185	2:192	α	16:247	1:383
3	n_{redun}	4:181	1:190	t_{succ}	18s:186	288s:197	t_{out}	720s:185	1800s:189	k	16:247	2:383
4	n_{repair}	5:181	1:186	n_{succ}	32:186	8:190	r_{group}	2:185	64:186	t_{stab}	1152s:247	288s:421
5							$n_{contact}$	16:185	2:278			

TABLE IX

THE RELATIVE IMPORTANCE OF EACH PARAMETER ON A DHT'S LATENCY VS. COST TRADEOFF.
Shows the same results as Table VIII, but using median lookup latency (in ms) as the performance metric.

worst columns show the failure rates and latencies achieved at an example bandwidth cost of 40 bytes/node/s, preceded by the values of that parameter that achieve those performances while setting *other* parameters *to their best values*. The data show the significance of the effect of tuning a particular parameter on the best failure rate and latency. For example, changing the Chord base b does not affect failure rate much, while changing b from 16 to 2 increases latency from 186 milliseconds to 226.

The following subsections explain the effect of tuning the parameters and the implications for protocol design; Table I summarizes the conclusions.

A. When To Use OneHop

Figures 4 and 5 show that OneHop has the best overall performance, with a failure rate close to 0% and a median latency of about 160 ms. The latter is close to the minimum possible latency imposed by the underlying simulated network's 156 ms median round trip time. A natural question is whether OneHop is therefore the best DHT protocol. The answer lies in the fact that OneHop's per-node bandwidth consumption is proportional to the churn rate and the number of nodes in the network. This makes OneHop primarily attractive in small or low-churn systems.

OneHop effectively has only one performance/cost tradeoff. Figure 5 shows that OneHop has a minimum bandwidth consumption at 7.5 bytes/node/s and it reaches reaches its best performance soon thereafter. OneHop cannot consume less bandwidth than this number because it always pro-actively notifies all nodes of *all* events. Therefore, OneHop's minimum bandwidth consumption scales linearly with churn rate and the size of the network. To demonstrate this property, we evaluated OneHop in a network of 3000 nodes[2] and show the results in Figure 6.

Figure 6 shows that the OneHop's minimum bandwidth consumption (the leftmost point of the OneHop curve) has a bandwidth cost of approximately 21 bytes/node/s for 3000-node networks. The threefold increase in the number of nodes triples the total number of join/leave events that must be delivered to every node in the network, causing OneHop to triple its bandwidth consumption from 7.5 to 21 bytes/node/s. For comparison, we also include Chord in Figure 6. The per-node state and lookup hop count of Chord scales as $O(\log n)$ and hence the convex hull of the 3000-node Chord network is shifted from the one for the 1024-node network by only a small amount towards the upper right. Therefore, for a 3000-node network, OneHop is only preferable to Chord when the

[2]As we do not have King data for our 3000 node topology, we derive our 3000-node pair-wise latencies from the distance between two random points in a Euclidean square. The median latency is the same as that of our 1024-node network.

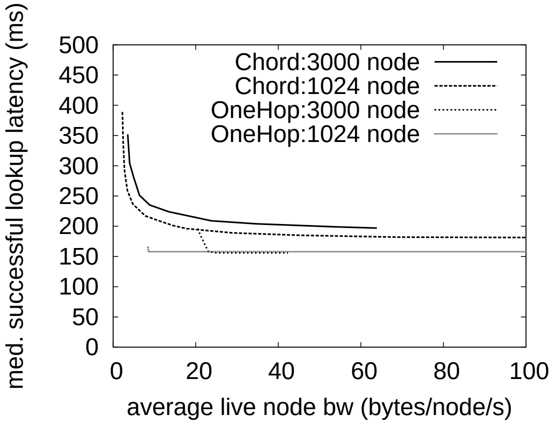


Fig. 6. Overall convex hulls for Chord and OneHop in 1024- and 3000-node networks, under churn intensive workload.

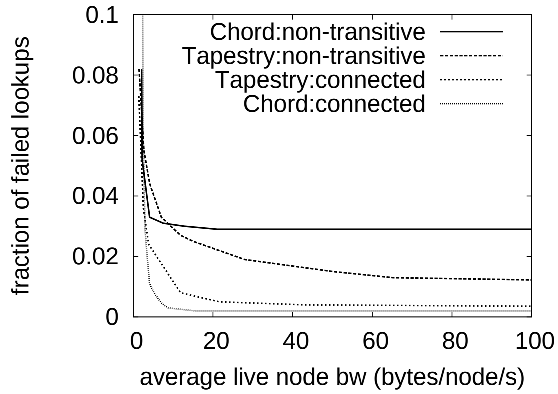


Fig. 7. Overall convex hulls for lookup failure rates for Chord and Tapestry under connected and non-transitive networks, under the churn intensive workload.

deployment scenario allows a communication cost greater than 20 bytes per node per second.

Another aspect of OneHop's performance is that slice and unit leaders use about 8 to 10 times more network bandwidth than the average. Chord, Kelips, Tapestry and Kademlia, on the other hand, have more uniform bandwidth consumption: the 95^{th}-percentile node uses no more than twice the average bandwidth. Therefore, if no node in the network can handle the bandwidth required of slice or unit leaders, one would prefer a symmetric protocol to OneHop.

Since OneHop's parameters don't allow significant tuning of the performance/cost tradeoff, we do not include this protocol in the rest of our analysis.

B. Separation of Lookup Correctness from Performance

Figure 4 shows that Chord provides a lower failure rate than the other protocols when the amount of bandwidth is limited. The following PVC parameter analysis explains why.

Table VIII shows that the most important Chord parameter, in terms of failure rates, is the successor stabilization interval (t_{succ}). This parameter governs how often a Chord node checks that its successor is still alive, and thus the amount of time it takes a Chord node to realize that its successor is dead and should be replaced with the next live node in ID space. The reason that t_{succ} has the largest effect on failure rate is that the correctness of the Chord lookup protocol depends only on successor pointers, and not on the rest of the Chord routing table [7]. Thus it is enough to stabilize only the successors frequently if a low lookup failure rate is required.

The other protocols do not have any similar entry in their routing tables that is sufficient for correctness; all routing information is equally important. Thus if one wishes a low lookup failure rate, the entire routing table must be stabilized frequently. The expense of this stabilization leads to a less attractive failure rate/bandwidth tradeoff at low bandwidths.

C. Coping with Non-transitive Networks

The flexibility and network-independence of the PVC framework also allows comparison of specific aspects of DHT protocols under anomalous network conditions. For example, DHT protocols typically have explicit provisions for handling node failure. These provisions usually handle network partitions in a reasonable way: the nodes in each partition agree with each other that they are alive, and agree that nodes in the other partitions are dead. Network failures that are not partitions are harder to handle, since they cause nodes to disagree on which nodes are alive. For example, if node A can reach B, and B can reach C, but A cannot reach C, then they will probably disagree on how to divide the key ID space among the nodes. A network that behaves in this manner is said to be non-transitive.

In order to measure the effects of this kind of network failure on DHTs, we created a topology exhibiting non-transitivity by discarding all packets between 5% of the node pairs in our standard 1024-node topology, in a manner consistent with the observed 4% of broken pairs [21] on PlanetLab [22]. The existence of PlanetLab nodes that can communicate on only one of Internet-1 and Internet-2, combined with nodes that can communicate on both networks, produces non-transitive connectivity between nodes. We ran both Chord and Tapestry churn intensive experiments using this topology, and measured the resulting failure rates of the protocols. Both protocols employ recursive lookup, and thus nodes always communicate with a relatively stable set of neighbors, eliminating the problem that occurs in iterative routing (*e.g.*, Kelips, Kademlia and OneHop) in which a node hears about a next hop from another node, but cannot communicate with that next hop.

We disable the standard join algorithms for both Chord and Tapestry in these tests, and replace them with an *oracle* algorithm that immediately and correctly initializes the state of all nodes in the network whenever a new node joins. Without this modification, nodes often fail to join at all in a non-transitive network. Our goal is to start by investigating the effect of non-transitivity on lookups, leaving the effect on join for future work. This modification changes the bandwidth consumption of the protocols, so these results are not directly comparable to Figure 4.

Figure 7 shows the effect of non-transitivity on the failure rates of Tapestry and Chord. Table X shows the parameter rankings and the performance range of different parameter

	Tapestry			Chord		
	param	**best**	**worst**	**param**	**best**	**worst**
1	t_{stab}	18s:0.013	1152s:0.075	t_{succ}	18s:0.029	288s:0.048
2	b	16:0.013	2:0.024	t_{finger}	18s:0.029	1152s:0.031
3	n_{redun}	4:0.013	1:0.023	n_{succ}	32:0.029	8:0.031
4	n_{repair}	1:0.013	10:0.014	b	2:0.029	128:0.035

TABLE X

THE RELATIVE IMPORTANCE OF CHORD/TAPESTRY PARAMETERS ON FAILURE RATE VS. COST TRADEOFF, NON-TRANSITIVE.

The **best** and **worst** parameter settings are for a fixed bandwidth budget of 80 bytes/node/s; at this cost most of the failures are due to network non-transitivity rather than churn.

settings under the non-transitive network. Figure 7 shows that Chord's failure rate increases more than Tapestry's with non-transitivity; we can use PVC parameter analysis to explain this behavior. Recall that, for the fully-connected network (see Table VIII), base was the *least* important parameter for Tapestry, in terms of failure rate. However, Table X shows that for the non-transitive network, base becomes a more important parameter, ranking second behind stabilization interval (which is still necessary to cope with churn). For Chord, however, base remains an unimportant parameter.

We can explain this phenomenon by examining the way in which the two protocols enforce the structure of their routing tables. The Chord lookup algorithm assumes that the ring structure of the network is correct. If a Chord node n_1 cannot talk to its correct successor n_2 but can talk to the next node n_3, then n_1 may return n_3 for lookups that really should have found n_2. This error can arise if network connectivity is broken between even a single node pair.

Tapestry's surrogate routing, on the other hand, allows for a degree of leniency during the last few hops of routing. Strict progress according to the prefix-matching distance metric is not well defined once the lookup reaches a node with the largest matching prefix in the network. This means that even if the most direct path to the owner of a key is broken due to non-transitivity, surrogate routing may find another, more circuitous, path to the owner. This option is not available in Chord's strict linked-list structure, which only allows keys to be approached from one direction around the ring. Tapestry does suffer some failures, however. If a lookup reaches a node that knows of no other nodes matching a prefix of the same size with the key as itself, it will declare itself the owner, despite the existence of an unreachable owner somewhere else in the network. A bigger base results in more nodes matching the key with the same largest matching prefix and hence gives more opportunity to surrogate routing to route around broken network connectivity.

In summary, while existing DHT designs are not specifically provisioned to cope with non-transitivity, some protocols are better at handling it than others. Future techniques to circumvent broken connectivity may be adapted from existing algorithms.

D. Extra State Is Better than Faster Stabilization

Table IX shows the relative importance of each protocol parameter for lookup latency. For both Tapestry and Chord,

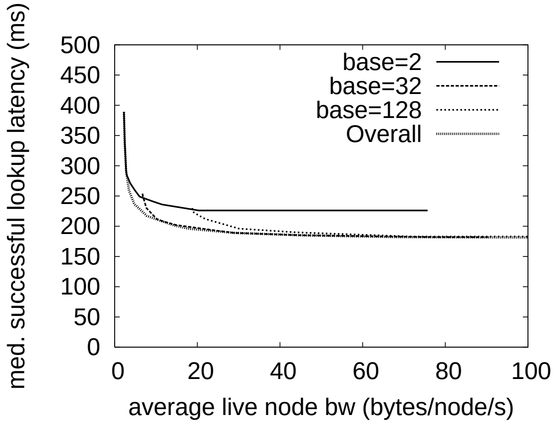


Fig. 8. Chord, under the churn intensive workload. Each line traces the convex hull of all experiments with a fixed base b value while varying all other parameters.

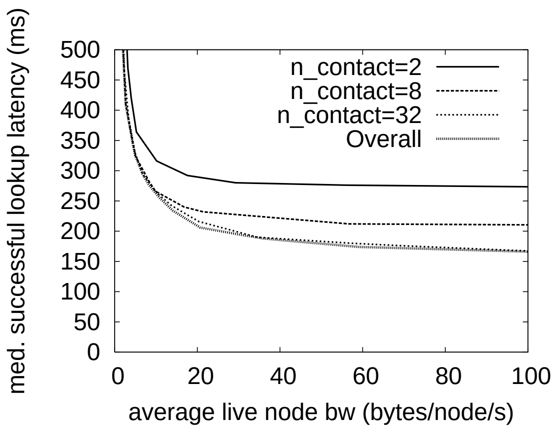


Fig. 9. Kelips, under the churn intensive workload. Each line traces the convex hull of all experiments with a fixed $n_{contact}$ value while varying all other parameters.

base (b) is the parameter whose value most needs to be tuned. Figure 8 shows the Chord parameter convex hulls for different base values. At the left side of the graph, where bandwidth consumption is small, smaller bases should be used to reduce stabilization communication cost. When more bandwidth can be consumed, larger bases lower the latency by decreasing the lookup hop-count.

A protocol could try to lower latency by stabilizing faster, rather than by increasing routing table size via the base parameter. Faster stabilization should decrease the likelihood of lookup timeouts. Tapestry and Chord would stabilize faster by decreasing the t_{stab} and t_{finger} parameters, respectively. The high importance rank of b in Table IX suggests that this approach would not be as efficient as increasing base. With a churn rate of 1 hour mean node lifetime, stabilizing to check neighbor liveness every 72s or 144s is sufficient to achieve a low lookup timeout probability for both Tapestry and Chord. Stabilizing faster to use additional bandwidth provides little extra benefit.

The Kelips parameter that controls the amount of allowed per-node state, $n_{contact}$, does not appear to be an important parameter. Figure 9 shows the parameter convex hulls for different values of $n_{contact}$ in Kelips. Large values of $n_{contact}$ (i.e., $n_{contact} > 16$) approach the overall convex hull performance, and thus tuning $n_{contact}$ (once a reasonable value has been chosen for some cost) affects the performance/cost tradeoff very little. This is because $n_{contact}$ only determines the amount of *allowed* state and the actual amount of state acquired by each node is determined by how fast nodes gossip (t_{gossip}) which is the most important parameter in Kelips.

Figure 9 shows that maximizing the amount of allowed state ($n_{contact} = 32$) achieves the best latency/cost tradeoff over the entire cost range for Kelips. In contrast, Figure 8 shows that one needs to tune the amount of allowed state by varying b in Chord to approach the overall hull. This difference between Kelips and Chord/Tapestry is due to differences in their routing structure. In Chord/Tapestry, base (b) not only determines the number of neighbors a node keeps, but also the part of the ID space in which these neighbors must lie. In contrast, the number of contacts a Kelips node keeps for each foreign group determines only the amount of allowed state, and not the ID space distribution of this state. Hence, it is always beneficial to allow as many contacts for each foreign group as possible as a particular contact is no more or less important than others.

Kelips' routing structure is flexible enough for each node to obtain complete state, therefore it achieves low latency comparable to OneHop with sufficient bandwidth consumption (Figure 5). However, unlike Chord/Tapestry, Kelips' routing structure is not flexible enough to allow routing state less than $\sqrt{n} + (\sqrt{n} - 1)$ as a node needs to keep $\sqrt{n}$ state for all nodes within a group and at least $\sqrt{n} - 1$ contacts, one for each foreign group. If the per-node routing state is less than this required minimum, Kelips' lookups have to go through randomly chosen nodes, resulting in significantly increased latencies. For this reason, Kelips' lookup latency is more than Chord's at low bandwidth in Figure 5.

E. Parallel Lookup Is More Efficient than Stabilization

Kademlia has the choice of using bandwidth for either stabilization or parallel lookups. Both approaches reduce the effect of timeouts: stabilization by eliminating stale routing table entries, and parallel lookups by overlapping activity on some paths with timeouts on others.

Table IX shows that stabilization is an inefficient way to reduce Kademlia's latency: the best value for t_{stab} is always the maximum stabilization interval. n_{tell} and α, which control the degree of lookup parallelism, are the most important parameters for latency. Larger values of α keep more lookups in flight, which decreases the likelihood that all progress is blocked by a timeout. Larger values of n_{tell} cause each lookup step to return more potential next hops and thus cause more opportunities for future parallelism. Thus, we conclude that parallel lookups reduce latency more efficiently than stabilization, under the churn intensive workload.

F. Learning from Lookups Can Replace Stabilization

Table VIII shows that Kademlia's t_{stab} parameter is no more effective at reducing failures than at reducing latency; again,

	Kademlia (no learn)		
	param	**best**	**worst**
1	t_{stab}	288s:0.036	1152s:0.122
2	α	4:0.036	1:0.096
3	n_{tell}	4:0.036	2:0.474
4	k	2:0.036	32:0.146

TABLE XI

THE RELATIVE IMPORTANCE OF KADEMLIA'S PARAMETERS ON FAILURE RATE VS. COST TRADEOFF, WITH LEARNING DISABLED. The best and worst parameter settings are for a fixed bandwidth budge of 80 bytes/node/s, as Kademlia without learning needs much more bandwidth to attain low failure rates.

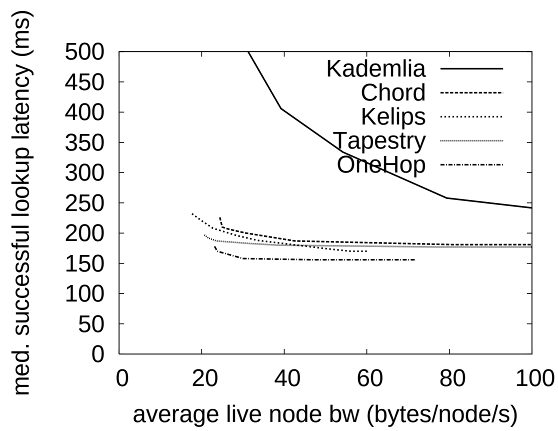


Fig. 10. Overall convex hulls for median successful lookup latency for all DHTs, under the lookup intensive workload.

the most efficient setting is the maximum stabilization interval. This suggests that Kademlia obtains information about new nodes through ways other than stabilization. Indeed, Kademlia relies on lookup traffic to learn about new neighbors: a node learns about up to n_{tell} new neighbors from each lookup hop. This turns out to be a more efficient way to keep routing tables up-to-date than explicit stabilization. For comparison, Table XI includes results from Kademlia with learning disabled. With no learning, tuning the stabilization interval is more important, and a faster stabilization rate is required for the lower lookup failures.

G. Effect of a Lookup-intensive Workload

The lookup intensive workload involves each node issuing a lookup request every 9 seconds, almost 67 times the rate of the churn intensive workload used in the preceding sections. As a result, the lookup traffic dominates the total bandwidth consumption. Figure 10 shows the overall latency convex hulls of all protocols under the lookup intensive workload.

PVC parameter analysis shows a decrease in the importance of tuning the Tapestry/Chord base parameter (b) with a lookup intensive workload. A large base (32 or 64) is the best for the lookup intensive workload under a wide range of bandwidth costs, since it reduces the number of lookup hops to approximately 2.8 one-way hops. Fewer hops translate into a large bandwidth decrease, given the large percentage of lookup traffic. For bases larger than 64, there is little reduction to the already-low average lookup hop-count. Therefore, latency reduction due to decreasing the number of timeouts becomes relatively more important, especially because the amount of stabilization traffic can be much less than the amount of lookup traffic. As a result, for both Chord and Tapestry, t_{stab} becomes the most important parameter to tune.

For Kademlia, α becomes the most important parameter to tune. Furthermore, a smaller α of 2 obtains the best tradeoff in the lookup intensive workload, as opposed to the larger α of 8 that optimizes latency for the churn intensive workload. Since Kademlia's stabilization process does not actively check the liveness of each routing table entry, stabilization is ineffective at reducing timeouts during lookups. Therefore, to achieve low lookup latency, some amount of lookup parallelism (i.e., $\alpha > 1$) is still needed. However, as lookup traffic dominates in the lookup intensive workload, lookup parallelism is quite expensive as it multiplies the already large amounts of lookup traffic. This partially explains why, in Figure 10, the overall convex hull of Kademlia suffers more from the lookup intensive workload than the other protocols.

VI. Related Work

This paper builds on previous DHT studies by explicitly accounting for the bandwidth consumed to achieve a certain lookup latency and providing a detailed comparison of multiple protocols using PVC. The comparison using PVC sheds light on which protocol features and parameters are important for achieving low latency under high churn.

Many existing DHT protocol proposals include performance evaluations [7]–[11], [23]. In general these evaluations have focused on hop-count and latency without churn, or the ability to maintain routing table correctness in the face of churn, but have rarely examined lookup performance in the face of churn. Similarly, most design studies explore tradeoffs in the context of static networks [1], [13], [24].

Liben-Nowell et al. [2] give a theoretical analysis of Chord in a network with churn. The concept of *half-life* is introduced to measure the rate of membership changes. It is shown that $\Omega(\log n)$ stabilization notifications are required per half-life to ensure efficient lookup with $O(\log n)$ hops. The analysis focuses only on the asymptotic communication cost due to Chord stabilization traffic, whereas our study explores a much broader set of parameters and protocols.

Rhea et al. [3] present Bamboo, a DHT protocol designed to handle networks with high churn efficiently and gracefully. Bamboo uses active probing with accurate TCP-like timeouts and specialized routing table stabilization strategies to perform well under churn. In a similar vein, Castro et al. [4] describe how they optimize their Pastry implementation, MSPastry, to handle consistent routing under churn with low overhead. Like these papers, our study is careful to separate detection of a failed node from recovering from failures during lookup.

Rhea et al. compare Bamboo against Pastry and Chord, but do not explore the parameter spaces of those protocols; our study explores a wider range of protocols and demonstrates that parameter tuning can have a large effect on performance. On the other hand, our simulator does not allow us to model bandwidth congestion, which proved to be an important factor in the comparisons done by Rhea et al.

Lam and Liu [5] present join and recovery algorithms for a hypercube-based DHT, and show through experimentation that their protocol gracefully handles both massive changes in network size and various rates of churn. While our work focuses on lookup latency and correctness, Lam and Liu explore K-consistency, a much stronger notion of network consistency that captures whether or not the network has knowledge of many alternate paths between nodes.

VII. Conclusions

Evaluating DHT protocols in the presence of churn is a challenge. Methodologies developed for static networks can be misleading, since they don't account for the resources consumed to obtain low latency. This paper introduces PVC, a performance vs. cost framework that explicitly accounts for the network bandwidth a DHT consumes to achieve better lookup performance.

DHTs incorporate many features to improve lookup performance at extra communication cost in the face of churn. It is misleading to evaluate the performance benefits of an individual design choice alone because other competing choices can be more efficient at using bandwidth. PVC presents designers with a methodology to determine the relative importance of tuning different protocol parameters under different workloads and network conditions. As parameters often control the extent to which a given protocol feature is enabled, PVC allows designers to judge whether a protocol feature is more efficient at using additional bandwidth than others via the analysis of the corresponding protocol parameters.

Using PVC and simulations of a set of DHT protocols with a wide range of design choices, we obtained a number of insights about protocol design, which are summarized in Table I. For example, PVC shows that to most efficiently use additional bandwidth, a DHT node needs to expand its routing table. Learning opportunistically can replace stabilization for acquiring new state.

We hope that other designers will find PVC useful in designing and evaluating new DHT features and protocols. The source code and simulation scenarios for this paper are available online at:
`http://pdos.lcs.mit.edu/p2psim`.

Acknowledgments

This research was conducted as part of the IRIS project (`http://project-iris.net`), supported by the National Science Foundation under Cooperative Agreement No. ANI-0225660.

We thank Frank Dabek for the King dataset measurements, and Russ Cox for his help writing the simulator. We also thank Steve Gerding for his contributions to our early non-transitivity analysis. Discussions with Sean Rhea, David Karger, and Frank Dabek substantially improved this work. Finally, we are grateful to Max Krohn and the anonymous reviewers for their insightful comments on previous drafts of this paper.

REFERENCES

[1] K. P. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of DHT routing geometry on resilience and proximity," in *Proceedings of the 2003 ACM SIGCOMM*, Aug. 2003.

[2] D. Liben-Nowell, H. Balakrishnan, and D. R. Karger, "Analysis of the evolution of peer-to-peer systems," in *Proceedings of the 21st PODC*, Aug. 2002.

[3] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT," in *Proceedings of the 2004 USENIX Technical Conference*, June 2004.

[4] M. Castro, M. Costa, and A. Rowstron, "Performance and dependability of structured peer-to-peer overlays," in *Proceedings of the 2004 DSN*, June 2004.

[5] S. S. Lam and H. Liu, "Failure recovery for structured P2P networks: Protocol design and performance evaluation," in *Proceedings of the 2004 SIGMETRICS*, June 2004.

[6] J. Li, J. Stribling, T. Gil, R. Morris, and M. F. Kaashoek, "Comparing the performance of distributed hash tables under churn," in *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems*, Feb. 2004.

[7] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Transactions on Networking*, pp. 149–160, 2002.

[8] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proceedings of the 1st IPTPS*, Mar. 2002.

[9] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse, "Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead," in *Proceedings of the 2nd IPTPS*, 2003.

[10] A. Gupta, B. Liskov, and R. Rodrigues, "Efficient routing for peer-to-peer overlays," in *Proceedings of the 1st NSDI*, Mar. 2004.

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

[12] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker, "Total recall: System support for automated availability management," in *Proceedings of the 1st NSDI*, Mar. 2004.

[13] F. Dabek, M. F. Kaashoek, J. Li, R. Morris, J. Robertson, and E. Sit, "Designing a DHT for low latency and high throughput," in *Proceedings of the 1st NSDI*, March 2004.

[14] S. Zhuang, I. Stoica, and R. Katz, "Exploring tradeoffs in failure detection in routing overlays," UC Berkeley, Computer Science Division, Tech. Rep. UCB/CSD-3-1285, Oct. 2003.

[15] A. Rodriguez, D. Kostic, and A. Vahdat, "Scalability in adaptive multi-metric overlays," in *Proceedings of the 24th ICDCS*, Mar. 2004.

[16] K. Hildrum, J. Kubiatowicz, S. Rao, and B. Zhao, "Distributed object location in a dynamic network," in *Proceedings of the 14th ACM SPAA*, Aug. 2002.

[17] D. 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 *Proceedings of the 29th STOC*, May 1997.

[18] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Nov. 2001.

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

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

[21] S. Gerding and J. Stribling, "Examining the tradeoffs of structured overlays in a dynamic non-transitive network," Class project: http://pdos.lcs.mit.edu/~strib/doc/networking_fall2003.ps, Dec. 2003.

[22] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating systems support for planetary-scale network services," in *Proceedings of the 1st NSDI*, March 2004.

[23] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Exploiting network proximity in peer-to-peer overlay networks," Microsoft Research, Tech. Rep. MSR-TR-2002-82, June 2002.

[24] J. Xu, "On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks," in *Proceedings of the 22nd Infocom*, Mar. 2003.