Examining the Tradeoffs of Structured Overlays in a Dynamic Non-Transitive Network

Steven Gerding, Jeremy Stribling
{sgerding, strib}@csail.mit.edu

Abstract

Although structured peer-to-peer overlays are an increasingly popular area of research, ranges of performance both within a single overlay and between multiple overlays have yet to be fully examined. In particular, the effects of adverse conditions found in real-world networks on overlay performance have not been adequately quantified. In this paper, we present and analyze data extracted from the PlanetLab network, focusing mainly on the presence of churn and non-transitivity, and use this data to drive simulations of several structured peer-to-peer overlays.

The protocols we use in our simulations are Chord, Tapestry, Kademlia, and Kelips. For each overlay, we simulate a range of different parameter settings, attempt to illustrate the tradeoffs inherent in parameter choices, and draw comparisons between the different protocols. We explore the performance of these overlays on a simulated PlanetLab topology, with and without the pathological conditions mentioned above. Our results indicate that non-transitivity can have a large effect on the performance of some overlays, while the amount of churn seen on PlanetLab tends to have a less significant impact.

1 Introduction

The performance of structured peer-to-peer overlays under real-world network conditions is a relatively unexplored area of research. Even fewer studies exist that compare the relative performance of several different peer-to-peer algorithms and examine their various performance tradeoffs. While recent research has focused on the performance of different peer-to-peer geometries on specific metrics (i.e. static resilience and local convergence) [6], or on examining high-level comparisons between the institution-specific implementations of a few overlays [17], there are several aspects of peer-to-peer performance that have not been addressed by this work.

For example, most peer-to-peer systems can expect a high degree of *churn* in its membership; that is, the rate at which nodes join and leave the network is likely to be high [7, 18]. In order for an overlay to remain connected under such conditions, each node must periodically probe for the liveness of its neighbors. The rate at which each node probes its neighbors incurs a cost in terms of bandwidth, proportional to the amount of state kept by the node. As bandwidth is often the limiting resource to an overlay, it is important to understand the tradeoffs overlays can make to minimize this maintenance bandwidth, while still achieving reasonable performance.

Another issue is how different peer-to-peer algorithms react to pathological conditions that arise in real-world networks. The impacts of conditions such as time-varying latencies, link failures, and non-transitive links on overlays is poorly understood at best, although we observe that these are all characteristics of a real-world distributed system testbed, PlanetLab [1, 16]. We believe that analyzing the ability of peer-to-peer algorithms to cope with such conditions, and understanding the tradeoffs of this ability on performance, will enable more robust overlay designs in the future.

In this paper, we explore and compare the impact of churn and other pathological network conditions on several different structured peer-to-peer overlay networks: Chord [19], Tapestry [21], Kademlia [15], and Kelips [9]. These overlays vary in their geometries and in the amount of state maintained, leading to differences in performance/efficiency tradeoffs. Furthermore, each algorithm has many different parameters that can be finely-tuned to improve performance and efficiency. We analyze the effect of this knob-turning for each overlay and explore the parameter space to find the best performance envelope for different network and churn conditions. We also compare the different overlays to each other, and discuss the tradeoffs a system designer can make when choosing, configuring, or designing an overlay. We quantify this performance/efficiency tradeoff by comparing bandwidth consumed per node (in bytes per second) to the average latency of a lookup (in milliseconds); this allows us to correctly account for background maintenance traffic as well as timeouts incurred during lookups due to stale routing data.

We perform our evaluations on p2psim,[1] a peer-to-peer simulator recently developed by the Parallel and Distributed Operating Systems group at MIT. The topologies we use for our simulations are extracted directly from latency and failure data gathered on PlanetLab [1]; another

[1] http://pdos.lcs.mit.edu/p2psim

contribution of this paper is the presentation and analysis of several different features of this data. Using real-world data in a simulator enables us to compare a non-trivial number of overlay networks under conditions that could actually arise in a deployment situation, without requiring the multi-institution collaboration, massive debugging effort, and extensive testing that would be necessary to gather the same data on PlanetLab itself. There are already several overlays and overlay services running continuously on PlanetLab, but often the set of participating nodes is hand-tuned by researchers to avoid nodes that exhibit strange network behavior. One goal of our work is to understand the effects this behavior has on overlays, enabling researchers to design systems that can easily run under a wide variety of network conditions.

The next section provides an overview of previous work in peer-to-peer overlay performance analysis and comparison, comparing and contrasting it to our approach. We follow this with a presentation of our PlanetLab data set in Section 3, and then give a brief overview of the four overlay protocols we explore in Section 4. We present and analyze our experimental results in Section 5, discuss their implications in Section 6, and conclude in Section 7.

2 Related Work

In the area of peer-to-peer system performance evaluation, there has been previous work focusing on topics related to the thesis of this paper. Various peer-to-peer overlay protocols proposed in the literature [4, 8, 9, 15, 19, 21] include performance evaluations in a static network. Lookup hop-count and latency are the usual metrics.

In [12], Liben-Nowell et. al. 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.

In [20], Xu studies the tradeoff of routing state versus network diameter. The study concludes that existing overlay protocols that maintain $\log(n)$ state have achieved the optimal asymptotic state vs. network diameter tradeoffs. Chord is asymptotically optimal, but further improvements to its exact state/diameter tradeoff are still possible.

Recent work by Gummadi et. al. [6] has begun to systematically compare the performance of different overlays and examine the impact of various design choices. In particular, Gummadi et. al. analyze how different routing geometries affect overlay resilience and proximity routing. However, their analysis is done without taking stabilization into consideration and therefore is unable to quantify the *cost* of different overlay protocols.

Rhea, Roscoe and Kubiatowicz compared the implementations of two overlays [17], focusing primarily on the case for benchmarks as a way to better understand different design decisions. Our work focuses on the behavior of the algorithms, using synthetic workloads to explore specific behavior aspects. We believe both approaches are necessary to derive a complete and robust understanding of peer-to-peer overlay networks.

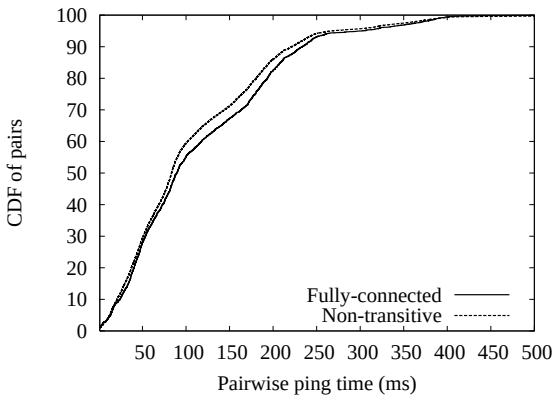

Figure 1: *Median pairwise latency distribution in PlanetLab.*

Other recent work [10, 13] proposes peer-to-peer overlays based on de Bruijn graphs as resilient alternatives to existing overlay protocols. While we feel this is a promising area of research, space and time constraints prevent us from including de Bruijn-based overlays in our experiments. We plan to explore this in future work. Similarly, unstructured and randomized overlay networks [3, 5, 14] are beyond the scope of this paper.

3 Data Set

In this section we present the characteristics of our PlanetLab-based topology, as extracted from continuous measurement of PlanetLab [1]. The data set consists of fully-pairwise ping times taken between all PlanetLab nodes, at intervals of fifteen minutes. Each data point is the average of ten pings attempts. If ten pings have not successfully returned within two minutes, the link is considered down. Our analysis focuses on measurements taken between November 1, 2003 and November 18, 2003.

To carry out our experiments, we derived two different topologies from the PlanetLab data set: a **fully-connected** topology and a **non-transitive** topology. The fully-connected topology consists of 159 PlanetLab nodes, each of which is directly connected to every other node in the set. This topology is used as a baseline in our comparisons. The non-transitive topology contains all of the 248 nodes in the PlanetLab data set,[2] and is used to simulate the real-world conditions that peer-to-peer overlays will be subjected to when put into production (see Section 3.2).

[2]Nodes which were observed to be dead for the duration of our data set were excluded.

The distribution of median RTTs between nodes in each topology is shown in Figure 1. The mean RTT for both topologies is approximately 118 ms.

We chose the PlanetLab data set for our study because it represents a large, global-scale testbed on which peer-to-peer systems are currently being deployed. However, these continuously-running systems do not fully utilize the entire set of available nodes, mostly because of pathological network conditions. This data set exhibits real-world levels of node failure, link non-transitivity, and time-varying link latency. Understanding the effect of these properties on overlay performance and robustness is the major goal of this work; these properties are discussed in the following three subsections.

3.1 Failures

The degree to which failures occur in the environment in which a peer-to-peer overlay network operates is a major determinant of its performance. Two important types of failures that a system must take into account are link failures and node failures (i.e. churn rate). A link failure occurs when a path through the network connecting two nodes becomes unusable and two formerly connected nodes can no longer communicate with each other. Link failures have a variety of different causes, including the presence of severe congestion in the network, or the failure of an intermediate router along the path. A node failure occurs when a particular node is cut off from the entire system, reboots or becomes deactivated for maintenance, or fails altogether.

In order to simulate conditions that accurately reflect real world networks, we sought to characterize the frequency of both node failures and link failures in our data set. A previous study of the PlanetLab testbed [2] shows that the mean time to failure (MTTF) of nodes in the system is 321.7 hours (approximately two weeks), and that the mean time to recovery (MTTR) from these failures is 2.5 hours. Our analysis of the testbed shows that the MTTF of links in PlanetLab is 9.48 hours and the MTTR from these failures is 2.69 hours.

It is easy to imagine the effects that failures might have on peer-to-peer overlay networks. One would expect an increase in the amount of resources devoted to stabilization and ultimately, if failure rates are sufficiently severe, the complete inability to route requests. We seek to quantify these effects using detailed simulation and real-world data. Currently, p2psim is equipped to simulate node churn, but is unable to model the effects of link failures; as such, link failures will not be included in our analysis. We do, however, feel that link failures are an important aspect of real-world network topologies and leave their exploration to future work.

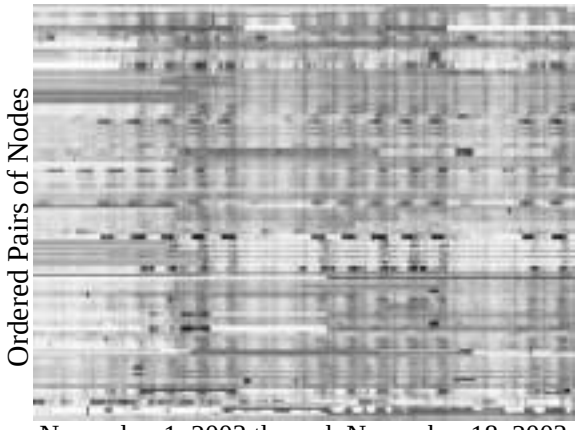

Figure 2: *Pairwise latency variation in PlanetLab over time.* Each horizontal line corresponds to an ordered pair of PlanetLab nodes, and the intensity of each point on that line denotes a relative ping time between the two nodes over the course of eighteen days. White represents the minimum ping time seen, black the maximum, and middle gray represents the median. We use a weighted contrast curve to scale intermediate colors, emphasizing variations near the median. See Section 3.3 for discussion.

3.2 Non-Transitivity

Another important characteristic of the PlanetLab data set is the presence of non-transitive links. A set of three nodes is said to exhibit non-transitivity if two of the nodes are unable to directly communicate with each other, even though they are both able to communicate with the third node. For example, given the set of nodes $\{A, B, C\}$, if A is able to communicate with B, B is able to communicate with C, but A is not able to directly communicate with C, the situation is said to be non-transitive. Note that any set of more than three nodes exhibiting non-transitivity must contain three nodes that meet this definition.

Non-transitivity could present problems for peer-to-peer overlay networks, depending on their methods of routing information dissemination. For example, systems that route requests iteratively rather than recursively may have problems when the next hop suggested by a neighbor is not reachable from the source of the query, but looks alive to the neighbor suggesting it.

The primary cause of non-transitivity in our data set is the existence of three distinct classes of nodes: Internet1-only nodes, Internet2-only nodes, and nodes that are multi-homed, i.e. able to communicate on both Internet1 and Internet2. Internet2-only nodes are not able to directly communicate with Internet1-only nodes and vice versa, while both are able to communicate with multi-homed nodes, leading to non-transitivity. The bulk of the nodes in the PlanetLab testbed are hosted at universities, which tend to be multi-homed; however, the system also incorporates many nodes that are hosted at corporate sites with a connection only to Internet1, as well as

nodes co-located with Internet2 routers. The result is that PlanetLab contains a nontrivial number of nodes from all three of the aforementioned classes. As the diversity of the network connections in the system continues to grow, it will become increasingly more important for peer-to-peer overlay networks deployed on PlanetLab to take non-transitivity into account.

To assess the degree of non-transitivity present in PlanetLab, we examined all possible combinations of three nodes from the 248 available in the data set. If exactly two of the three links between the nodes in a combination were functioning, the combination was declared to be non-transitive. Of the $\frac{248!}{3! \times 245!} = 5022992$ combinations analyzed, 248536 (approximately 9.9%) were non-transitive. Based on this observation, it is clear that non-transitivity is a non-negligible feature of our data set.

3.3 Time-Varying Latency

PlanetLab's network environment is not isolated, nor is its workload static. As such, we expect the latency between nodes to vary with time as the network undergoes different levels of congestion and the load on the nodes fluctuates. Any overlay deployed in such an environment must cope with a constantly changing topology, and overlays that depend on locality properties for their performance must make continual estimates of round trip times in the network and update routing information as necessary.

In Figure 2, we plot the variation in ping times between pairs of nodes in our data set, relative to the median ping time of the pair. The intensity of each point on the graph indicates the relative ping time between two PlanetLab nodes for one measurement, plotted over eighteen days in November of 2003. Overall ping times tend to stay relatively close to their medians (represented by the middle gray color); white corresponds to the minimum ping time seen between the two nodes, while black is the maximum. We see by the vertical bands in the graph that ping times tend to fluctuate in aggregate, and in fact correspond with well-known time-of-day network usage patterns. Close inspection reveals that there are exactly eighteen thin, evenly-spaced darker vertical bands, occurring approximately between noon and 9:00 pm Eastern time each day. Moreover, there are two darker sets of thick vertical bands, each made up of five thin bands, corresponding to the two full Monday to Friday weeks covered by this data set. On average, the ping time between two nodes can by vary by a full second from the median, and in some cases by more than thirty seconds.

4 Overlays

This paper compares four peer-to-peer overlay protocols: Chord [19], Tapestry [21], Kademlia [15] and Kelips [9]. The protocols have been implemented in the simulator in as much detail as possible (by the authors and others), based on published papers and communication with the architects of each overlay. The protocols have been somewhat adapted in order to implement a compatible lookup function, which takes a key as an argument, and pursues the lookup until it has contacted the node responsible for that key. This section briefly summarizes the protocols and discusses the parameters that we varied in order to explore protocol performance. [3]

Number of successors	4 – 32
Finger base	2 – 128
Finger stabilization interval	2 min – 32 min
Succlist stabilization interval	1 min – 32 min

Table 1: *Chord parameters.*

Base	2 - 128
Stabilization interval	2 min – 32 min
Number of backup nodes	1 – 4
Backup nodes used in a lookup	1 – 4

Table 2: *Tapestry parameters.*

4.1 Chord

In Chord, for our simulations, a lookup completes when it reaches the node whose ID most closely precedes the key (the *predecessor*), which can provide the ID of the key's successor. The base b of the ID space is variable for this implementation: a node with ID x keeps $(b-1)\log_b(n)$ fingers (used for *performance*) whose IDs lie at exponentially increasing fractions of the ID space away from itself. Any node whose ID lies within the range $x+(\frac{b-1}{b})^{i+1}*2^{160}$ and $x+(\frac{b-1}{b})^{i}*2^{160}$, modulo 2^{160}, can be used as the i^{th} finger of x. This flexibility allows for Proximity Neighbor Selection [6]. Each node also keeps a *successor list* of s nodes (used for *correctness*). Chord can route either iteratively or recursively (see [19] for explanations of these different styles); we explore both.

A Chord node x periodically pings all current fingers to check their liveness. If a finger i doesn't respond, x issues a lookup request for the key $x + (\frac{b-1}{b})^{i} * 2^{160}$, yielding node f. Node x retrieves f's successor list, and uses the successor with the lowest latency as the level i finger. A node stabilizes its successor list by periodically retrieving and merging its successor's successor list. Table 1 lists the Chord parameters that are varied in the simulations.

4.2 Tapestry

A Tapestry node ID can be viewed as a sequence of l base-b digits. A routing table has l levels, each with b entries. Nodes in the m^{th} level table share a prefix of length $m-1$ digits, but differ in the m^{th} digit. Each entry may contain up to c nodes, sorted by latency. The closest

[3] A subset of these descriptions first appeared in a recent submission [11], and was written collectively by those authors.

Nodes per entry (k)	8, 16, 32
Parallel lookups (α)	1 - 5
Stabilization timer	2 min - 32 min
Refresh timer	2 min - 32 min

Table 3: *Kademlia parameters.*

of these nodes is the entry's *primary neighbor*; the others serve as *backup neighbors*.

Nodes forward a lookup message for a key by resolving successive digits in the 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 (see [21] for the details of this algorithm). Routing in Tapestry is recursive.

For lookups to be correct, at least one neighbor in each routing table entry must be alive. Tapestry periodically checks the liveness of each primary neighbor, and if the node is found to be dead, the next closest backup in that entry (if one exists) becomes the primary. If a primary is found to be dead in the course of a lookup (i.e. an RPC to that node timed out), a backup may be used instead; the number of backups used in such a way is a configurable parameter in our implementation. Table 2 lists the parameters varied in the simulations.

4.3 Kademlia

A Kademlia node's routing table consists of b buckets, where b is the number of bits in a Kademlia node ID. Bucket i in node x's routing table consists of (at most) k node IDs that share the first i most significant bits in x's node ID. In a bucket, node IDs are sorted by the last time they have been seen, with the least-recently seen node at the head.

A node x does a lookup for key A by sending parallel RPCs to the α nodes whose IDs are closest to A, using XOR as the distance metric. A node replies to this RPC by sending back a list of the k nodes it believes are closest to A. Node x then sends RPCs to these new nodes, trying at all times to keep α outstanding RPCs. The lookup converges in O(log n) iterations to the node responsible for A, (i.e. the node whose ID is closest to A). In our tests, nodes only look up values corresponding to node IDs; the lookup succeeds as soon as communication with that node is established.

A node updates its routing table on every incoming RPC request or reply, updating the most-recently-seen timestamp. Every stabilization period each node checks its routing state, and if all entries in a bucket are older than the value of the refresh timer, the node refreshes its routing table by doing a lookup for a key in the bucket's range. Thus, Kademlia can leverage communication due to lookups to stabilize its routing state, without requiring extra background traffic. Table 3 summarizes the parameters we vary in the Kademlia simulations.

Gossip interval	0.125 min – 24 min
Group targets	1 – 64
Contact targets	1 – 16
Group ration	1 – 64
Contact ration	1 – 16
Contacts per group	2 – 8
Times a new item is gossiped	0 – 4
Routing entry timeout	5 min – 40 min

Table 4: *Kelips parameters.*

4.4 Kelips

Kelips nodes divide themselves into k groups, where k is chosen to be the square root of the number of nodes. A node's group is its ID mod k. Each node's routing table contains an entry for every other node in its group, and "contact" entries for a few nodes from each of the other groups. Thus a node's routing table size is a small constant times $\sqrt{n}$, in a network with n nodes.

In the simulator's variant of Kelips, lookups are only defined for node IDs. The originating node executes a lookup for a key by asking a contact in the key's group for the IP address of the target key's node, and then (iteratively) contacting that node. If that fails, the originator tries routing the lookup through randomly chosen nodes.

Nodes periodically gossip to discover the existence of new members of the network, and may also learn about other nodes due to lookup communication. Routing table entries that have not been refreshed for a certain period of time expire. Nodes learn RTTs and liveness information from each RPC, and preferentially route lookups through low-RTT contacts.

Table 4 lists the parameters we vary for Kelips. Targets are the number of randomly chosen nodes a node sends information to every gossip interval, and rations are the number of nodes mentioned in the gossip messages. Contacts per group is the maximum number of contact entries per group in a node's routing table; if it has value c, then the size of each node's routing table is $\sqrt{n} + c(\sqrt{n} - 1)$.

5 Evaluation

In this section we present the results of simulating the four overlay protocols described in Section 4, preceded by a description of our experimental setup and methodology. We first give baseline results for all overlays in a static fully-connected environment, and then show the effects of churn and non-transitivity on performance, both separately and combined. Finally, we explore the effects of time-varying latencies on lookups.

5.1 Experimental Setup

We performed the experiments presented in this section using p2psim, a recently-developed discrete event simulator designed for peer-to-peer overlay networks. Currently p2psim models only network latency, not bandwidth con-

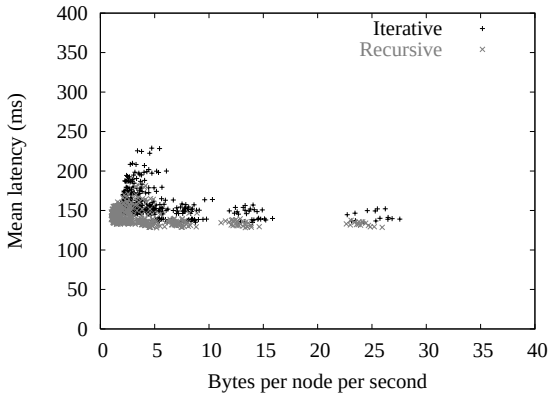

Figure 3: *Tradeoffs in Chord under the baseline scenario.* Shown for both iterative and recursive routing. The finger base and stabilization intervals have the greatest effect on performance.

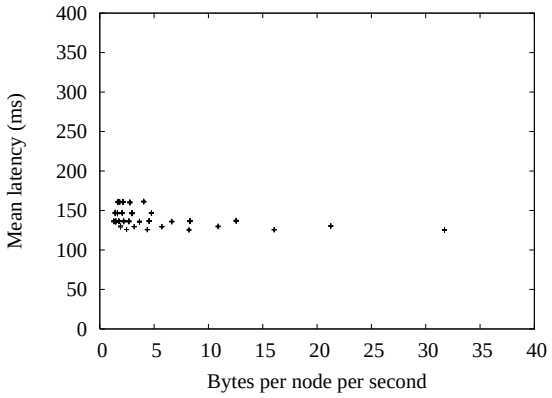

Figure 4: *Tradeoffs in Tapestry under the baseline scenario.* The ID base and stabilization intervals have the most effect on performance. Fewer points are shown due to a smaller parameter range.

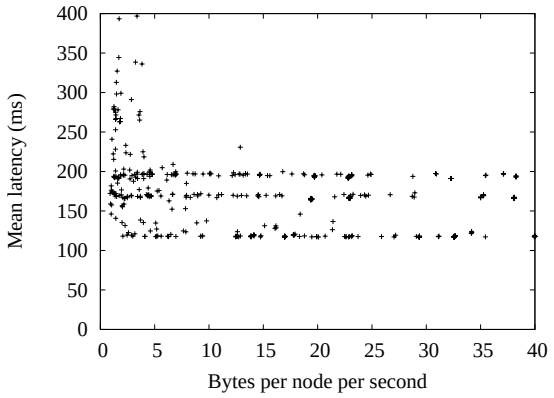

Figure 5: *Tradeoffs in Kelips under the baseline scenario.* The number of known contacts per group and the gossip interval have the greatest effect on performance.

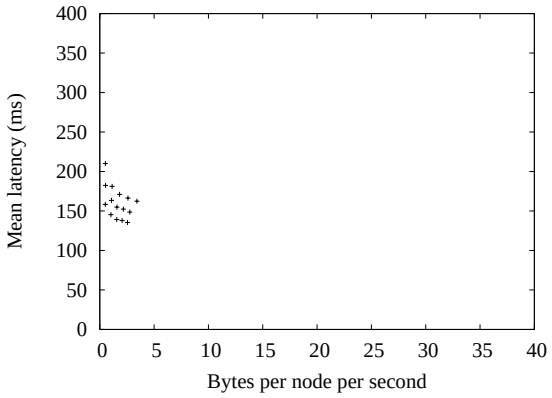

Figure 6: *Tradeoffs in Kademlia under the baseline scenario.* Due to its use of lookups for stabilization, Kademlia uses very little bandwidth. Both k and α have an effect on performance.

straints or queuing delay, which suffices for comparing the relative performance of different parameter configurations and protocols. All topologies used in our experiments were generated from our PlanetLab data set (Section 3). As a workload, each live node in the network generates lookups for random keys in the identifier space. The time between lookups issued by an individual node is exponentially distributed about a mean of 116 seconds, the mean time between requests measured by a recent study of an unstructured peer-to-peer system [18]. This ensures that each node generates several lookups per discrete sample in our data set. The length of each simulation is six hours in simulated time, unless otherwise noted. All overlays begin the test with complete, ideal routing information.

The metric by which we evaluate the performance of these four overlays illustrates the tradeoff between two important aspects of peer-to-peer systems: total bandwidth usage and lookup latency. Bandwidth is often a serious bottleneck in such systems, and minimizing the bandwidth consumed per node (due to lookup and stabilization traffic) is a very real goal for system architects. On the other hand, finding data as quickly as possible is important for the user experience. Hence, our simulations examine the tradeoff between bandwidth and latency to see which parameter settings and overlays offer the best performance in terms of both these metrics. Each graph in this section plots bandwidth consumed (in bytes per node per second) versus the average latency of a lookup (in milliseconds); each point on the graph represents a full experiment run with a certain set of parameters. We optimistically model timeouts experienced when communicating with a dead node as one round-trip time, and these timeouts are accounted for when computing the mean lookup latencies in the following sections. Furthermore, lookup

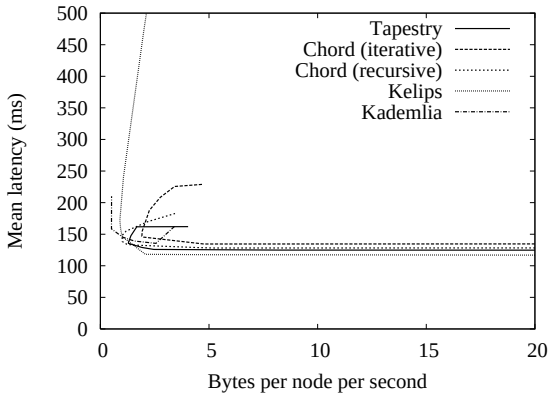

Figure 7: *The performance tradeoffs of all overlays in the baseline scenario.* The curve shown is the convex hull of the mass of points for each protocol, representing the optimal performance tradeoff for that protocol.

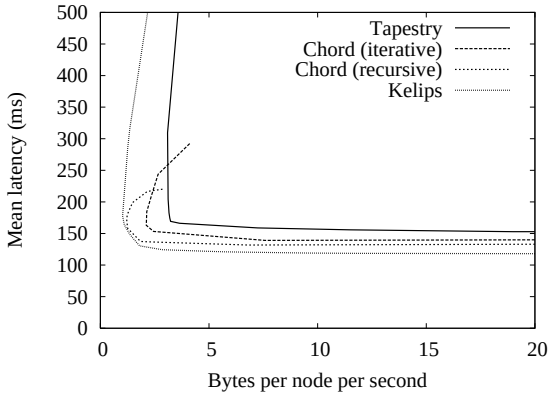

Figure 8: *The performance tradeoffs of all overlays in the churn scenario.*

results are checked for correctness and queries are retried if their results are incorrect, increasing lookup latency.

5.2 Baseline

As a baseline performance metric, we ran a set of experiments for each protocol on the fully-connected topology of 159 nodes with no churn, using the median latency observed between each pair of nodes. Not surprisingly, each protocol performs best when it keeps a large amount of state and stabilizes as infrequently as possible. Figures 3, 4, 5 and 6 demonstrate the effect of different parameter settings on Chord, Tapestry, Kelips, and Kademlia, respectively. Maintaining a higher number of neighbors (the base parameters in Chord and Tapestry, contacts per group in Kelips, and number of entries per bucket in Kademlia) lowers the latency, while decreasing the amount of stabilization traffic in Chord, Tapestry and Kelips lowers the bandwidth. Because Kademlia uses lookup traffic to stabilize, it does not need to use extra bandwidth for stabilization; this accounts for both its low bandwidth numbers as well as the seeming lack of points, as many parameter settings appear as points with the same value. Changing the number of parallel lookups in Kademlia is the main source of increasing bandwidth, lowering lookup latencies at the same time.

To illustrate tradeoffs between overlays, we present the results for all protocols in this baseline environment in Figure 7. Each line on the graph represents the *convex hull* of the mass of data points collected for each overlay; essentially, this is the curve that traces the best performance tradeoffs available to an individual overlay. At the knees of the curves, where both latency and bandwidth are small, we see that in this baseline scenario all protocols behave similarly, though the two-hop scheme of Kelips has a slightly lower latency than the other overlays.

5.3 Churn

Next, we wish to explore the degree to which the rate of churn seen on PlanetLab affects the different protocols. As mentioned in Section 3.1, PlanetLab nodes have mean lifetimes of about two weeks, and a mean time to repair of about 2.5 hours; we scaled these numbers down linearly to fit into the time scale of our simulations. We ran these simulations on the fully-connected topology of 159 nodes. Unfortunately, the p2psim implementation of Kademlia is too CPU-intensive to run with the churn scenario, so we do not include Kademlia numbers in this section or Section 5.5.

Due to space constraints, we omit the full data point graphs for individual overlays, and simply show the convex hull performances for all overlays under churn in Figure 8. Because the rate of churn is so low on PlanetLab, the performance tradeoffs of the overlays are relatively unaffected. A notable exception is Tapestry: the figure shows a clear increase in both bandwidth and latency from the static case. We attribute this to the fact that Tapestry does not separate performance from correctness in its routing state, and thus stabilizes all neighbors at the same rate, leading to greater bandwidth usage. Similarly, it cannot afford to ping neighbors as frequently, and thus suffers more timeouts due to stale routing state.

5.4 Non-Transitivity

To examine how these overlays perform in the presence of non-transitive links, we simulated them in our non-transitive, 248-node topology with no churn (see Section 3.2). A node that cannot communicate over a particular link sees the link as a three second timeout, chosen to reflect transport level timeout values for new connections.

Figure 9 shows the convex hull performance of each overlay in the presence of non-transitivity. Perhaps surprisingly, we find that non-transitivity does not dramatically affect Tapestry and Kelips when compared to the baseline scenario, while Chord (both iterative *and* recursive) sees a tremendous increase in latency. Kademlia sees

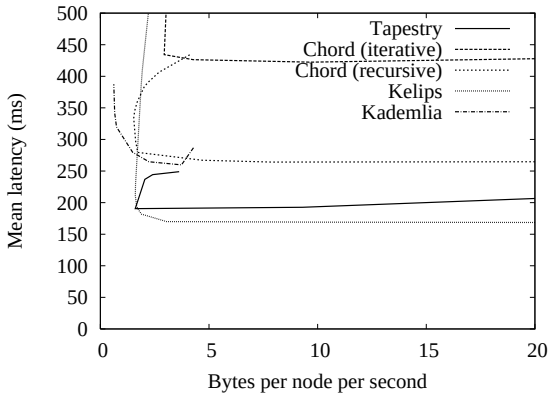

Figure 9: *The performance tradeoffs of all overlays in the non-transitive scenario.*

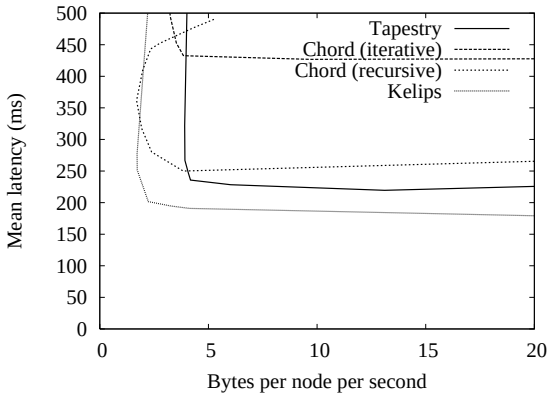

Figure 10: *The performance tradeoffs of all overlays in the non-transitive scenario with churn.*

a moderate latency jump, as well.

The large performance gap between Tapestry and Chord can be explained by two important observations. First, Tapestry's use of recursive routing ensures that it communicates with the same set of neighbors throughout the duration of the test, and thus after one round of stabilization will remove all unreachable nodes from its routing table. In contrast, a node using iterative routing is constantly forced to communicate with new nodes during lookups; in the presence of non-transitivity this will lead to costly timeouts. However, this does not explain why Tapestry also performs better than recursive Chord in this scenario – this is due to the strictness of Chord routing, when compared to the less restrictive style of Tapestry routing. Chord may only route in one direction around the ID space, regardless of the routing style in use. Therefore, if routing reaches a node where the only links available to a certain node point past the successor (even if other nodes in the network can reach the successor), the query will incorrectly terminate prematurely. In Tapestry, however, a lookup may proceed past the destination and then travel back in the opposite direction (due to its surrogate routing algorithm) until it eventually finds the correct destination. Kelips, which has the least restrictive routing algorithm, does even better than Tapestry under non-transitivity.

5.5 Churn and Non-Transitivity

Figure 10 shows the results of our simulations for Tapestry, Chord, and Kelips on the 248-node non-transitive topology, with churn. Again, the introduction of churn does not significantly affect the performance of any of the overlays. Mainly we see that Tapestry suffers an increase in latency and bandwidth, just as in the fully-connected churn scenario.

5.6 Time-Varying Latency

Finally, we tested the effect of time-varying latencies on performance, as discussed in Section 3.3. We present the results for Tapestry in Figure 11. This test was conducted in the fully-connected topology, with no churn, and lasted for three days of simulated time. Variable latency links seem to have little effect on performance when compared to the baseline scenario in Figure 4; the only difference is a slight increase in lookup latency. Looking again at Figure 2, we see that the latencies between all pairs of nodes tend to fluctuate in diurnal patterns, all at the same time. Therefore, even though the latency of a particular link may have increased, so have the latencies of all other links, so there is no room for improvement. Figure 2 also shows that the mean ping times of links tend to be above their median ping times, hence the increase in average lookup latency. Space and time constraints prevent us from presenting time-varying latency results for all overlays, but the analysis of the Tapestry results lead us to suspect there will be no major impact on performance.

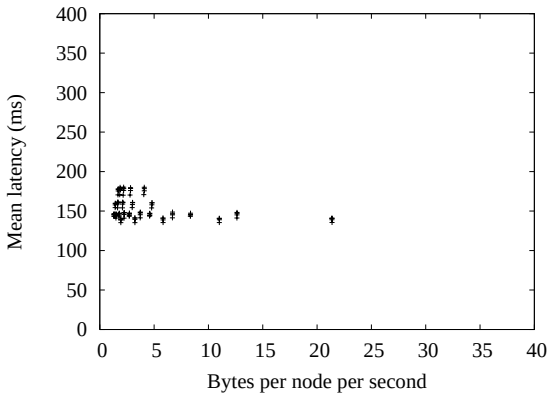

Figure 11: *Tradeoffs in Tapestry under the time-varying latency scenario.*

6 Discussion

Analysis of the data obtained from our experiments has taught us several general lessons relating the design of

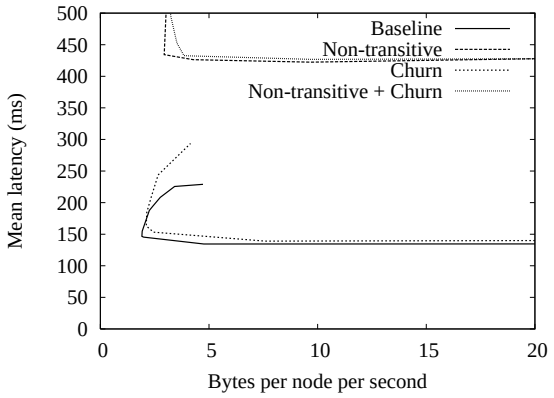

Figure 12: *Tradeoffs in Chord (iterative) under all conditions.*

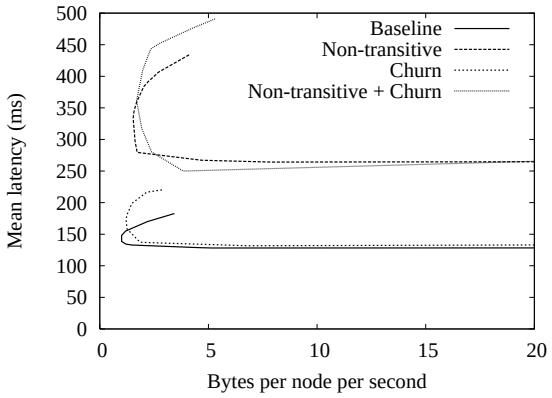

Figure 13: *Tradeoffs in Chord (recursive) under all conditions.*

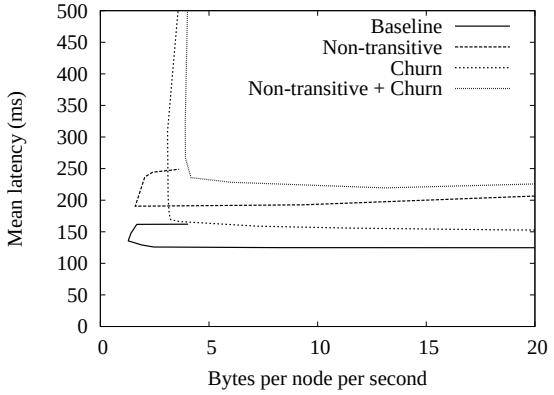

Figure 14: *Tradeoffs in Tapestry under all conditions.*

various aspects of peer-to-peer overlays to their effects on performance. We feel that these lessons are important to designers of future structured overlays, and present them in this section.

Parameter Tuning: In examining Figures 3, 4, 5 and 6, we see a wide range of performance over the spectrum of parameters for each overlay. The amount of bandwidth consumed varies by a factor of 30 in most of the overlays, and in the case of Kelips, the mean lookup latency varies by a factor of almost 4. This drastic variation in performance indicates that parameter tuning is an extremely important part of implementing peer-to-peer overlays in a real world application. In addition, the optimal set of parameters may change as the underlying network conditions evolve. For this reason, an area of future work in overlay design could be to explore the benefits of dynamically self-tuning parameters. Our study has shown that a general heuristic to follow in choosing overlay parameters for a network such as PlanetLab is to maintain a large number of neighbors (corresponding to a high base or number of contacts per group) and minimize maintenance traffic (corresponding to a large stabilization interval). We hypothesize that these configurations perform well because of the low degree of node churn present in the PlanetLab network.

Churn: Previous work has emphasized the importance of churn rates on the performance of overlay systems; however, our study has shown that the real-world rates of churn observed on PlanetLab do not have a significant effect on the performance of peer-to-peer overlays. Figures 12 and 13 show the performance of iterative and recursive Chord in each of our experiments. For both graphs, the baseline and churn curves are very close together, as are the non-transitive and non-transitive + churn curves. This indicates that churn has a negligible effect on the performance of Chord, which is consistent with most of the other overlays. The one exception to this trend is the performance of Tapestry, as shown in Figure 14. We have discussed in Section 5.3 that this is primarily due to the fact that Tapestry does not separate performance state from correctness state in its routing tables. In other, more transient networks, higher rates of churn may have a more significant effect on the performance of peer-to-peer overlays and the separation of performance from correctness could become an imperative design objective.

Non-Transitivity: Our study has shown that the presence of link non-transitivity in a network can have a considerable impact on the performance of peer-to-peer overlays. In examining Figures 12 and 13, we can see the large discrepancy between Chord's performance with and without non-transitive links. In contrast, however, we can see from Figure 14 that non-transitivity has less of an effect on the performance of Tapestry. As presented in Section 5.4, this is because Tapestry uses only recursive routing and is able to travel both forward and backward in the identifier space in search of a key. Although the use of

recursive routing forces a node to relinquish fine-grained control over the routing process (e.g. the ability to abort a lookup in progress), it is generally more efficient than iterative routing, and much more robust to deleterious network conditions such as non-transitivity. In addition, if the degree of non-transitivity is significant in the target environment of a peer-to-peer overlay, as it is in PlanetLab, the ability to traverse the identifier space in both directions may be crucial to the performance of the overlay.

7 Conclusions and Future Work

We have explored the performance and robustness of four structured peer-to-peer overlays in the presence of network characteristics culled from the PlanetLab data set. Our results show that while the amount of churn experienced by PlanetLab nodes does not have a significant impact on most overlays, the presence of non-transitivity in a network can greatly impair the ability of certain overlay designs to route quickly and efficiently. To our knowledge, the effects of non-transitivity has not been explored in previous overlay research, and we have found it can be an important aspect of the underlying network layer.

One obvious direction for future work in this area is to expand and diversify the types of overlays examined; for example, studying overlays at different points in the state/efficiency tradeoff space (e.g. Koorde [10] or unstructured overlays). We would also like to explore the effects of other real-world pathological network conditions, such as link failures, packet loss rate, and asymmetric link latencies. Furthermore, examining how scaling the number of nodes beyond a few hundred affects performance, as well as determining methods for allowing self-tuning parameters, would be interesting areas of future research.

The goal of this work has not been to find the "best" overlay, but rather to explore the interaction between overlay properties and the underlying network conditions. We hope that this work will help inform the decisions of future systems designers, leading to the deployment of more efficient, robust, and scalable overlays on dynamic, non-transitive networks.

Acknowledgments

We would like to thank Jinyang Li, Thomer M. Gil, Frans Kaashoek, Robert Morris, and Frank Dabek for their work on the simulator and overlay implementations used in this paper, as well as for numerous useful discussions. The authors of [2] provided valuable insight into how to render Figure 2, and Frank Dabek and Jinyang Li helped us fix its contrast levels. Finally, thanks to Hari Balakrishnan and the 6.829 TAs for their instruction and help over the past semester.

References

[1] PlanetLab All-Pairs-Pings. http://pdos.lcs.mit.edu/∼strib/pl app/.

[2] ANONYMOUS. Workload and failure characterization on a large-scale federated testbed. Under submission to SIGMETRICS 2004.

[3] CASTRO, M., COSTA, M., AND ROWSTRON, A. Should we build Gnutella on a structured overlay? In *Proc. of HotNets-II* (November 2003).

[4] CASTRO, M., DRUSCHEL, P., HU, Y. C., AND ROWSTRON, A. Exploiting network proximity in peer-to-peer overlay networks. Tech. Rep. MSR-TR-2002-82, Microsoft Research, 2002.

[5] CHAWATHE, Y., RATNASAMY, S., BRESLAU, L., LANHAM, N., AND SHENKER, S. Making Gnutella-like P2P systems scalable. In *Proc. of SIGCOMM* (August 2003).

[6] GUMMADI, K., GUMMADI, R., GRIBBLE, S., RATNASAMY, S., SHENKER, S., AND STOICA, I. The impact of DHT routing geometry on resilience and proximity. In *Proc. of SIGCOMM* (August 2003).

[7] GUMMADI, K. P., DUNN, R. J., SAROIU, S., GRIBBLE, S. D., LEVY, H. M., AND J.ZAHORJAN. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. of SOSP* (October 2003).

[8] GUPTA, A., LISKOV, B., AND RODRIGUES, R. One hop lookups for peer-to-peer overlays. In *Proc. of Hot Topics in Operating Systems* (May 2003).

[9] GUPTA, I., BIRMAN, K., LINGA, P., DEMERS, A., AND VAN RENESSE, R. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proc. of IPTPS* (February 2003).

[10] KAASHOEK, F., AND KARGER, D. Koorde: A simple degree-optimal hash table. In *Proc. of IPTPS* (February 2003).

[11] LI, J., STRIBLING, J., MORRIS, R., KAASHOEK, M. F., AND GIL, T. M. DHT routing tradeoffs in networks with churn. Under submission to IPTPS 2004.

[12] LIBEN-NOWELL, D., BALAKRISHNAN, H., AND KARGER, D. R. Analysis of the evolution of peer-to-peer systems. In *Proc. of Symposium on Principles of Distributed Computing* (August 2002).

[13] LOGUINOV, D., KUMAR, A., RAI, V., AND GANESH, S. Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience. In *Proc. of SIGCOMM* (August 2003).

[14] MANKU, G. S. Routing networks for distributed hash tables. In *Proc. of Symposium on Principles of Distributed Computing* (July 2003).

[15] MAYMOUNKOV, P., AND MAZIERES, D. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. of IPTPS* (March 2002).

[16] PETERSON, L., ANDERSON, T., CULLER, D., AND ROSCOE, T. A blueprint for introducing disruptive technology into the internet. In *Proc. of HotNets-I* (October 2002).

[17] RHEA, S., ROSCOE, T., AND KUBIATOWICZ, J. Structured peer-to-peer overlays need application-driven benchmarks. In *Proc. of IPTPS* (February 2003).

[18] SAROIU, S., GUMMADI, K. P., DUNN, R. J., GRIBBLE, S. D., AND LEVY, H. M. An analysis of internet content delivery systems. In *Proc. of OSDI* (December 2002).

[19] STOICA, I., MORRIS, R., LIBEN-NOWELL, D., KARGER, D., KAASHOEK, M. F., DABEK, F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking* (2002), 149–160.

[20] XU, J. On the Fundamental Tradeoffs between Routing Table Size and Network Diameter in Peer-to-Peer Networks. In *Proc. of Infocom* (March 2003).

[21] ZHAO, B. Y., HUANG, L., STRIBLING, J., RHEA, S. C., JOSEPH, A. D., AND KUBIATOWICZ, J. D. Tapestry: A global-scale overlay for rapid service deployment. *IEEE Journal on Selected Areas in Communications* (2003). Special Issue on Service Overlay Networks, to appear.