Apocrypha: Making P2P Overlays Network-aware

Prasanna Ganesan Qixiang Sun Hector Garcia-Molina
Stanford University, Stanford, CA 94305, USA
{prasannag, qsun, hector}@cs.stanford.edu

Abstract— **Many P2P systems organize nodes in an overlay network, in order to enable communication between nodes. We devise a generic mechanism called Apocrypha that can optimize a variety of overlay networks with respect to the underlying physical-network structure. In particular, Apocrypha is applicable to deterministic and randomized overlay topologies, which offer optimal degree-diameter trade-offs but are not amenable to common overlay-optimization techniques proposed in prior work. We show that the use of Apocrypha can lead to optimized overlays of higher quality than was possible with earlier work. We discuss how Apocrypha operates under high rates of system churn, with nodes joining and leaving all the time, by trading off optimization cost against the benefit obtainable at high churn rates.**

I. Introduction

Many peer-to-peer systems, ranging from file-sharing systems and distributed hash tables (DHTs) to cooperative caching and multicast systems, are being developed on top of overlay networks. Different P2P systems utilize overlay networks in different ways; systems like Gnutella [1] use it to support a flooding primitive, while DHTs use the overlay to *route* messages from one node to another.

A fundamental problem confronting the design of all overlay networks is to ensure that communication on the overlay is as efficient as possible. The metric govering efficiency is, in general, dependent on the application; for example, an application that uses the overlay for routing may want to minimize the latency of the routing path between every pair of nodes, while another application may to minimize the average latency of all overlay links. A general rule of thumb, however, is that the overlay network needs to be constructed so as to "resemble" the underlying physical network, with neighbors on the overlay network also being "close" to each other in the physical network.

In this paper, we introduce a generic mechanism called **Apocrypha**[1] to improve the quality of overlay networks, which is applicable to a broad class of overlay networks used in different applications. Overlay topologies can be classified into three categories: deterministic, non-deterministic and randomized. (We present formal definitions shortly.) It has been observed that the most "desirable" topologies, in terms of graph-theoretic properties, are either deterministic or randomized [2], [3]. On the other hand, prior techniques for overlay optimization [4], [5], [6] are applicable only to non-deterministic topologies, leaving open the issue of how to optimize the deterministic and randomized topologies.

A primary contribution of Apocrypha is in filling the void – being applicable to all three classes of topologies – and helping construct optimized versions of the "desirable" deterministic and randomized overlays. We show that the use of Apocrypha on deterministic and randomized topologies can lead to optimized overlays of a quality comparable to, or better than, non-deterministic overlays optimized by earlier techniques.

A second contribution of Apocrypha lies in the manner in which it handles continuous node joins and leaves on the overlay network (system *churn*). As nodes join and leave, the overlay network changes, requiring continuous re-optimization of the overlay. However, there is a trade-off involved: the higher the rate of churn in the network, the more expensive it is to maintain a high-quality network. Apocrypha exposes this trade-off to limit the cost of optimization, settling for a lower-quality network in periods of high churn.

Organization In Section II, we define the different classes of overlay topologies, explaining their differences and motivating the need for Apocrypha. In Section III, we describe Apocrypha as an abstract, centralized algorithm to optimize a static overlay network. We apply this algorithm to a variety of overlay networks, and compare its fundamental power to that of other approaches for optimizing overlays. We evaluate Apocrypha using both physical-network models, and measurements on the PlanetLab [7] system. Section IV describes how to develop Apocrypha as a low-cost, distributed algorithm for optimizing overlays. Section V discusses how Apocrypha trades off overlay quality for optimization cost by adapting to the rate of churn in the system. We discuss related work in Section VI and conclude

[1]A P2P Overlay ConstRuction to Yield Proximal neigHbors (Allegedly)

in Section VII. Appendix I discusses different routing protocols on Chord that we introduced and used in our evaluation. Appendix II discusses the evaluation of Apocrypha on Gnutella, and an interesting phenomenon called *long circuits* that emerges in such flooding-based networks.

II. Classifying Overlay Topologies

We begin by defining the notion of an overlay topology. Each node in an overlay network is associated with a label, called the node ID, which is usually selected uniformly at random [8], [4]. An *overlay topology* is a set of rules that, given a set of node IDs as input, dictate which pairs of IDs should have a link between each other in the overlay network. Overlay topologies are classified into three groups:

Deterministic overlay topologies are those where the set of links is a deterministic function of the IDs, i.e., given the set of IDs, the set of links between IDs is fully determined. Examples of deterministic topologies include Chord [8], CAN [9], and emulations of the de Bruijn graph such as Koorde [2] and D2B [10].

Non-deterministic topologies are those where there are many possible valid interconnections for a given set of IDs. Examples include Pastry [11], Tapestry [12] and non-deterministic variants of Chord [6], [5]. Non-deterministic topologies are characterized by *choice*: a node with a given ID has a choice *that it may exercise arbitrarily* in determining what IDs it will form links with.

Randomized topologies are those where the set of links between IDs are a *randomized* function of the set of IDs. Examples include Symphony [13], Viceroy [14] and low-diameter random graphs [15]. Note the key difference from non-deterministic topologies: although there are any number of links that may be chosen as part of the network, the choice needs to be exercised by a specified random process. In contrast, non-deterministic topologies allow the choice to be exercised arbitrarily.

The nature of the topology has interesting implications for the graph properties of the resulting overlay network. Deterministic topologies based on de Bruijn graphs [2], [10] as well as some randomized topologies [14], [3] offer asymptotically optimal trade-offs between graph degree and routing path length, i.e., for the limited number of links allowed per node in these overlay networks, the number of hops used for routing is asymptotically optimal. In contrast, none of the non-deterministic topologies offer optimal routing hops.

Prior work on optimizing overlays [4], [5], [6] has focused on non-deterministic topologies, therefore being limited to optimizing the latency of routing for each hop, but having no power to reduce the number of hops themselves to an optimal value. A key question left open by these approaches is: *Is it possible to retain the optimal number of hops offered by well-known deterministic and randomized topologies, while simultaneously ensuring that the latency of each hop, and that of the routing path, is also optimized?*

Apocrypha answers this question in the affirmative, being applicable to all topologies in a uniform fashion. The generality of Apocrypha helps us investigate a natural follow-up question: *Given that all kinds of topologies can now be optimized for the physical network, what topology offers the best routing latency for a given graph degree? Can deterministic or randomized topologies be optimized well enough to outperform non-deterministic topologies?*

From our investigation in Section III, we determine that some deterministic and randomized topologies can indeed provide better routing performance than non-deterministic topologies, while other deterministic topologies based on emulations of de Bruijn graphs do not perform as well.

III. Optimizing Static Overlays : A Holistic View

We begin by considering the problem of optimizing an arbitrary *static* overlay network. Our focus is on centralized algorithms that have a "holistic" view of the system, i.e., know the entire overlay network as well as the properties of the underlying physical network, and use this holistic view in optimizing the overlay network. Our goal is to understand the power of different optimization techniques, and the fundamental limits imposed by different topologies on optimization. In Section IV, we will study how such optimization can be performed in practice, when nodes have only a local view of the network.

We begin with a simple observation: Given a fixed set of IDs to be allocated to nodes, and a topology (deterministic, non-deterministic or randomized) that the overlay network needs to conform to, there are a large number of different overlays that may be constructed by *assigning IDs to nodes in different fashions*. To illustrate, consider a set of four nodes A, B, C and D, with each node assigned a unique two-bit ID, with the overlay network required to be a "hypercube topology". Figure 1 shows three possible overlay networks that may be constructed on these four nodes (with the IDs of nodes in parentheses).

Observe that these three overlay networks are different in quality. For example, say nodes A and B are very

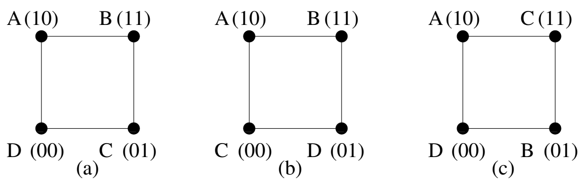

Fig. 1. Three different overlay networks for the 4-node hypercube topology

close on the physical network while C and D are far away from both A and B as well as from each other. Then, overlay network (a) or (b) would be more desirable than (c), since both (a) and (b) have a fast overlay link between A and B.

Using the above observation, we may now formally define the problem of optimizing a given overlay network: *Given (a) an initial overlay network N with each node assigned a unique ID, and (b) a blackbox function d to measure "distance" between any two nodes, create a new overlay network N' by re-assigning the original set of IDs among the nodes (and retaining overlay links between IDs as in N), such that the average distance, computed over all overlay links using d, is minimized.*

Note that the new overlay N' is constrained to have the same set of links between IDs as the original overlay N, thus ensuring that the overlay topology is preserved. Also note that we have chosen the average distance between all nodes connected by overlay links, as the metric to measure the "goodness" of an overlay network. From our experiments, we find that this metric serves as a good proxy for a variety of different application metrics.

The problem formulated above is NP-hard, being a more general form of the graph isomorphism problem, and we use a hill-climbing approach for optimization. To illustrate with our example in Figure 1, say the P2P system initially constructs overlay network (c). We can then transform (c) into (a) by a simple operation: swapping the IDs of nodes B and C, which corresponds to node B breaking its overlay link with node D and connecting to node A instead, and node C performing the reverse operation.

The hill-climbing algorithm simply performs a series of swaps, each of which helps improve the quality of the network, and is sketched in Algorithm 1. The term $nbrs(i)$ is shorthand for the set of neighbors of node i on the overlay network. Note that evaluating whether a swap would improve the quality metric or not is easy, since a swap only changes a small number of overlay links in the graph (see lines 4-5).

Our evaluation in this section will attempt to answer the following questions:

- How well does the hill-climbing algorithm help in optimizing different overlay topologies?

Algorithm 1 HillClimbing(OverlayNetwork N, DistanceFunction d)

1: **while** not converged **do**
2: **for** each node i **do**
3: Pick a random node j.
4: InitialCost= $\sum_{m \in nbrs(i)} d(i,m) + \sum_{n \in nbrs(j)} d(j,n)$;
5: NewCost= $\sum_{m \in nbrs(i))} d(j,m) + \sum_{n \in nbrs(j)} d(i,n)$;
6: **if** NewCost < InitialCost **then**
7: Swap IDs of i and j;
8: Eliminate original links of i and j.
9: Link i to the old $nbrs(j)$ and vice versa.
10: **end if**
11: **end for**
12: **end while**

- How does hill-climbing compare to alternative approaches for topology optimization?
- Of the different topologies that were optimized, which ones provide the best results with the use of hill-climbing?

To answer these questions, we consider different overlay topologies, and routing protocols, that we discuss below. We then explain alternative approaches for optimizing these topologies. Finally, we discuss our evaluation and answers to the above questions.

A. Overlay Topologies

We consider a number of different overlay topologies in our experiments.

Deterministic Topologies Our set of deterministic topologies includes *Chord* [8], and two different emulations of the de Bruijn graph, *Koorde* [2] and *D2B* [10]. We present a definition of Chord for completeness: All node IDs are drawn from the circular N-bit ID space $[0, 2^N)$; a node with ID x forms links to the nodes whose IDs immediately succeed the value $x + 2^i$ for $0 \le i < N$. When node IDs are chosen at random (as we assume is the case), the expected degree of each node is $O(\log n)$ in an n-node Chord network.

We refer the reader to [2], [10] for the definitions of the de Bruijn variants. We simply note two details here: Both Koorde and D2B may be constructed in a variety of "bases", with higher base values leading to higher graph degree and fewer routing hops between nodes. The expected degree of each node in a base-k Koorde/D2B is $O(k)$, although the constant factors hidden by the O-notation are different for the two topologies.

Nondeterministic Topologies For non-deterministic topologies, we consider the Chord variant proposed in [5], [6], which is defined as follows: For each $1 \le i < N$, a node with ID x forms a link to *any* node with an ID in the interval $[x+2^{i-1}, x+2^i)$. In addition, the node always maintains a link to the node with the next larger ID (its *successor*). To distinguish this topology from Chord, we refer to it as *Fuzzy Chord*. Fuzzy Chord also has a node degree of $O(\log n)$.

Randomized Topologies We consider a randomized version of Chord [3] defined as follows: For each $1 \leq i < N$, a node with ID x forms a link to the node whose ID immediately succeeds $x + 2^{i-1} + r_i$, where r_i is a number drawn uniformly at random from the interval $[0, 2^{i-1})$. In addition, each node always maintains a link to its successor. We refer to this topology as *R-Chord*. The node degree in R-Chord is also $O(\log n)$.

B. Routing Protocols

We consider three different routing protocols used on the above overlay topologies.

Greedy routing is applicable to all the topologies (although the protocol details are different for the de Bruijn variants compared to the Chord variants). On Chord, Fuzzy Chord and R-Chord, this protocol ensures routing in $O(\log n)$ hops, and close to $0.5 \log n$ hops in expectation [8], [3]. On the de Bruijn variants in base k, routing requires $O(\log n / \log k)$ hops, with the constant factors being different for the two variants. Note that greedy routing *is completely oblivious to* the pairwise distances between nodes and does not attempt to use the knowledge of inter-node distances to optimize the route in any way.

NoN-Greedy routing [3] applies to R-Chord, and uses additional information gleaned by "lookahead" to enable routing on R-Chord in $O(\log n / \log\log n)$ hops. Note that this routing protocol is also oblivious to pairwise-distance information.

Ebola is a routing protocol devised by us, which applies to Chord, Fuzzy Chord and R-Chord. Ebola uses lookahead information, just like NoN-Greedy, but additionally also exploits pairwise-distance information in route selection, to identify routes that not only have a small number of hops, but are also optimized for the *distance* of the routing path. (Such protocols are labeled PRS in [5].) We present a detailed description of Ebola in Appendix I. We merely state here that Ebola guarantees that routing requires only $O(\log n)$ hops, while heuristically optimizing for the distance of the routing path. Note that Ebola does not apply to the de Bruijn variants.

C. PNS: An Alternative Approach to Optimization

We now describe an alternative to hill-climbing, often labeled PNS [5], that has been studied in past literature [4], [5], [6]. PNS applies solely to non-deterministic topologies, and cannot be used on deterministic or randomized topologies. We present it with respect to Fuzzy Chord.

Recall that in Fuzzy Chord, a node with ID x forms its i^{th} link to any node with an ID in the interval $[x+2^{i-1}, x+2^i)$. PNS uses the following approach to optimize this link: Consider the set S_i of all nodes in this interval, and allow x to connect to the *closest* node, in terms of the blackbox distance, in S. This optimization may be applied to each link in the network, and we term the overlay network obtained in the fashion as *Fuzzy Chord (Perfect)*.

Finding the closest node in each interval may be "expensive" in a real system, leading to the consideration of a weaker form of PNS [5], [6], where a node merely *samples* k nodes from each interval, and links to the closest of the sampled nodes. We term the overlay network constructed with this approach *Fuzzy Chord (k)*.

Note that, in either case, routing on the overlay network may be performed using either the Greedy protocol or Ebola.

D. Evaluation

We now evaluate our Hill-climbing Algorithm 1 on the different topologies, and compare it to PNS. Our evaluation requires a matrix of "distances" between every pair of nodes, that the blackbox distance function can supply to the hill-climbing algorithm. We generate this matrix in two different ways.

GT-ITM: We use the GT-ITM [16] topology generator to generate transit-stub models of the underlying physical network. We assign latencies of 5ms, 20ms, and 100ms respectively to stub-stub, stub-transit and transit-transit links. Each overlay node is attached to a random stub-node, and we compute the distance between two overlay nodes as being the latency of the shortest path between their corresponding stub nodes in the GT-ITM graph. We experimented with different GT-ITM graphs with varying proportions of transit and stub domains to produce different matrices of inter-node latencies, and report results on a representative model. (Henceforth, we use the terms *distance* and *latency* interchangeably.)

PlanetLab: We also obtain all pairwise ping times [17] over a set of 182 nodes distributed across 128 sites spread around the world on the PlanetLab network [7]. We use these ping times as the estimate of pairwise distance for a 182-node overlay network.

1) Convergence of Hill-climbing: We first study the convergence of the hill-climbing algorithm for different overlay topologies. For each topology, we start with an overlay network of N nodes (N=25000 for GT-ITM

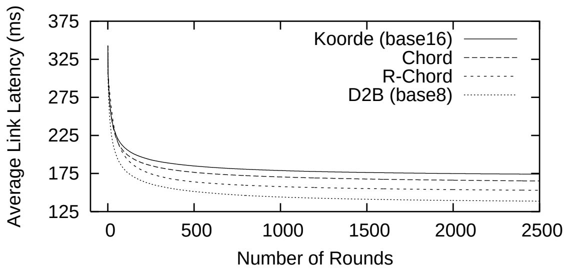

Fig. 2. Convergence of Hillclimbing (GT-ITM)

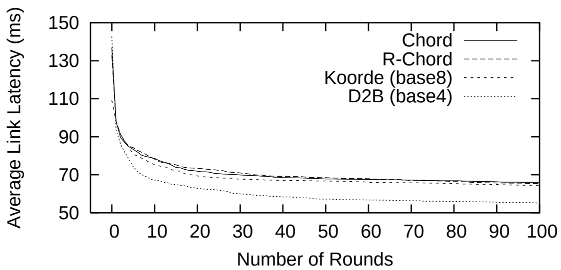

Fig. 3. Convergence of Hillclimbing on PlanetLab

experiments, and is 182 for PlanetLab) with nodes being assigned IDs independent of the underlying physical network. Koorde and D2B are constructed in bases chosen to ensure all topologies have nearly the same graph degree. (We do not report results on Fuzzy Chord, since there is no reason to use Fuzzy Chord instead of Chord or R-Chord, when optimizing with hill-climbing. We will apply PNS to Fuzzy Chord, however, later in our evaluation.)

We use both the GT-ITM model and PlanetLab data to create different matrices of inter-node latencies. Each hill-climbing step corresponds to an iteration of the *while*-loop in Algorithm 1, and may induce some pairs of nodes to swap their IDs.

Link Latency Figure 2 plots the average of the overlay-link latencies as a function of the number of hill-climbing steps for different topologies on the GT-ITM data. The initial average latency is around 340ms, corresponding to the average latency between pairs of nodes in our GT-ITM model. Within a few hundred steps, hill-climbing converges on all the different topologies and reduces the latency to roughly half the initial value. We discuss the question of whether the converged state represents a local or a global optimum in the next subsection.

We also repeat the experiment on PlanetLab data, and plot the results in Figure 3. Observe that the initial average latency is only 133ms, and the latency after optimization is again roughly half this initial value. We see that convergence on PlanetLab occurs much faster than on GT-ITM, within 30 steps, which is not surprising given that our overlay networks on PlanetLab consist of much fewer nodes.

Path Latency Recall that average link latency is not

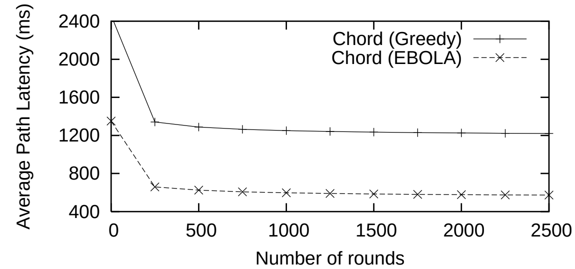

Fig. 4. Average Path Latency on Chord with different routing protocols

necessarily the primary metric of interest for the application. Figure 4 shows the routing *path latency* for Chord, using different routing protocols, as a function of the number of hill-climbing steps. The results are reported on the GT-ITM topology, and we also omit the evolution of path latencies for other topologies, which exhibit a similar behaviour. Observe that path latency, using both greedy routing and Ebola, exhibits the same convergence as the average link latency; most of the improvement is achieved within the first 200 steps, and the path latency quickly converges to the final value. We also see that Ebola outperforms greedy routing considerably, which is not surprising since Ebola can adapt to the actual latencies of various links in order to find efficient paths.

2) The Optimality of Hill-climbing: Our hill-climbing algorithm can only guarantee convergence to a local minimum. We would therefore like to obtain a lower bound on the quality of the optimal solution, so that we understand how close the algorithm's solution is to the optimal solution. Since obtaining a tight lower bound on average link latency is hard, we focus on a different quantity. For a pair of nodes i and j, define the latency of the shortest (in terms of latency, not number of hops) overlay path between i and j to be $SOPd(i,j)$. We are interested in the average of all such latencies $AllSOPd = \sum_{i,j \in V} SOPd(i,j)/|V|^2$.

This quantity has a simple lower bound which can be derived solely from the distance function d. If we assume that the triangle inequality is valid for d (which it is in our physical-network models), $SOPd(i,j)$ can be no less than the physical-network latency between i and j[2]. Thus, $INTd = \sum_{i,j \in V} d(i,j)/|V|^2$ is a lower bound on $AllSOPd$.

Figure 5 shows the evolution of $AllSOPd$ as a function of the number of steps taken by the algorithm, using the GT-ITM model of the physical network. The horizontal line titled "Internet" represents the average

[2]The triangle inequality is known to be invalid on the internet which may inflate our lower bounds somewhat. However, we do not believe this to be a critical problem, especially with overlay networks where each overlay hop needs to go through a tail link and an end host.

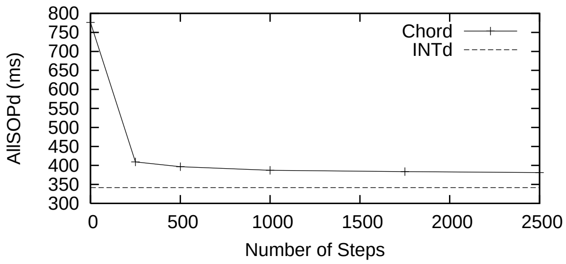

Fig. 5. Avg. Latency of the Shortest Overlay Path

physical-network latency between two nodes $INTd$, and is thus a lower bound on $AllSOPd$ for any overlay network. Once again, we see that we converge very quickly towards an optimal state. We note that the ratio of $AllSOPd$ to $INTd$ after 2500 steps is 1.11 for Chord, leading us to conjecture that the solutions produced by our algorithm are close to optimal, even for other metrics of interest.

3) Performance of Different Topologies: Table I depicts the average routing path latency for different topologies, using different routing protocols, both on the unoptimized overlays and on the optimized networks obtained by running Hill-climbing to convergence. The top half describes results on GT-ITM, while the bottom half applies to PlanetLab. We discuss each of these in turn.

GT-ITM On the unoptimized networks, Greedy routing on D2B and NoN-Greedy routing on R-Chord perform very well. This is not surprising since both of them use very few *hops* in routing. Greedy routing on Koorde does not perform as well because, although the number of routing hops is *asymptotically* optimal, the constant factors involved reduce its performance[3]. Surprisingly, the use of Ebola on Chord performs the best of all; Ebola uses more routing hops, but its awareness of different hop latencies enables it to find fast paths anyway.

After optimization, we observe that Ebola applied to Chord continues to perform the best, with an average path latency of only 574ms. NoN-Greedy routing on R-Chord comes in second, with an average latency of 674ms, showing that Apocrypha does help in reducing path latency even for topologies with an optimal number of hops. D2B and Koorde also experience large improvements in their routing performance, but still do not manage to out-perform Chord or R-Chord, suggesting that the latter two topologies are more amenable to optimization.

PlanetLab The results on PlanetLab generally mirror the results observed in GT-ITM with a few caveats. Since many nodes in PlanetLab are physically "close" to each other, and the routing paths are very short on our small-sized overlay network, it becomes harder to distinguish between the different routing protocols. However, we still observe that Ebola on Chord performs the best, trailed by R-Chord and D2B.

Mechanism	Topology	GT-ITM	PlanetLab
Hill-climbing	Chord	574	132
Hill-climbing	R-Chord	694	134
PNS	Fuzzy Chord (Perfect)	770	224
PNS	Fuzzy Chord (16)	853	219

TABLE II
COMPARING HILL-CLIMBING TO PNS ON AVERAGE PATH LATENCY (IN MS) USING BEST ROUTING PROTOCOL

4) Hill-climbing vs. PNS: We now compare the effectiveness of Hill-climbing against the alternative technique for optimization: PNS. In order to perform this comparison, we apply PNS to Fuzzy Chord, selecting the closest node in each interval to obtain *Fuzzy Chord (Perfect)*, as described earlier. We also test the weaker form of PNS that samples only 16 nodes per interval to obtain *Fuzzy Chord (16)*. As discussed earlier, PNS does not apply to the other topologies that we consider.

We evaluate the routing path latency on the two Fuzzy Chord topologies, using both Greedy routing and Ebola. Table II reports the path latency obtained using the *best* routing protocol for the two variants of Fuzzy Chord, as well as for hill-climbing-optimized versions of Chord and R-Chord. As we see from the table, both Chord and R-Chord offer lower average path latencies compared to even the perfect version of Fuzzy Chord[4]. This suggests that hill-climbing is a powerful approach to overlay optimization, enabling high-quality embeddings of overlays that are not possible with the sole use of PNS.

IV. Optimizing Static Overlays: A Distributed Solution

Having seen that the centralized hill-climbing approach helps optimize a variety of different overlay topologies, we consider how to develop a low-cost, distributed algorithm for optimization in a P2P system. As the first step, we continue to focus on a *static* overlay network, with no nodes joining and leaving the system. In Section V, we will discuss issues arising from node dynamism.

A. A Distributed Algorithm

One simple way to design a distributed algorithm to improve a (static) overlay network, is to just let each

[3]It is possible to enhance Koorde, allowing each node to link to many of its successors, which improves routing latency considerably. We do not consider this variant further in this paper.

[4]On PlanetLab, sampling 16 nodes slightly outperforms perfect sampling. This appears to be a consequence of the small size of the PlanetLab network and the absence of the triangle inequality.

GT-ITM	Koorde (base 16)	D2B (base 8)	Chord		R-Chord		
	Greedy	Greedy	Greedy	Ebola	Greedy	Ebola	NoN-Greedy
Unoptimized	5361	1550	2462	1351	2368	2136	1663
Optimized	2027	993	1220	574	1060	923	694

PlanetLab	Koorde (base 8)	D2B (base 4)	Chord		R-Chord		
	Greedy	Greedy	Greedy	Ebola	Greedy	Ebola	NoN-Greedy
Unoptimized	644	434	535	228	456	245	291
Optimized	414	204	227	132	192	134	152

TABLE I

EFFECT OF HILL-CLIMBING ON AVERAGE PATH LATENCY (MS) FOR DIFFERENT TOPOLOGIES

node execute steps 3-12 of Algorithm 1 in a periodic fashion. There are three major questions arising from such an approach, and we will answer these questions in turn.

- How can a node find another random node j in step 3 of the algorithm?
- How can a pair of nodes decide if a swap is beneficial or not?
- How can the algorithm's overhead be minimized, given that it needs to run continuously?

1) Finding a Random Node: To find a random node j in a distributed manner as in step 3 of Algorithm 1, each node i initiates a *probe*, a random walk originating at itself and proceeding for W steps, and selects the destination of the walk as j, the candidate for the swap. Most overlay topologies that we consider have good expansion properties; consequently, setting W to half the diameter of the overlay network ensures that the random walk leads to a random node. Note that these probes can often be piggybacked on normal overlay traffic, thus reducing the overhead of initiating probes.

Biasing probes: Instead of using a purely random walk to find a candidate, we could "bias" the random walk to seek candidates more likely to swap places with the probe initiator. The policy we consider is to assign every node a *dissatisfaction index* which is equal to the average of its overlay-link latencies. A probe continues to be a random walk, but returns the most dissatisfied node encountered on the walk, instead of just returning the termination point. The danger in using such a policy is that the likelihood of converging to a local minimum is higher than with random hill climbing. Our experiments in this section evaluate whether such biasing is useful.

2) To Swap or Not to Swap: Once a node i finds a candidate node j to potentially swap with, the two nodes need to collaborate and determine whether a swap is beneficial to the overlay (lines 4-5 of Algorithm 1). In order to perform this determination, i needs to compute its distance from each of j's neighbors, and vice versa, by invoking the "blackbox" distance function. The blackbox may be implemented in different ways. One approach is to use ICMP pings to enable each node to measure distances from itself. Another approach is to associate Euclidean vectors with each node, as suggested in [18], [19] with the distance between vectors serving as an estimate of actual network distance.

In either case, nodes i and j can compute all the necessary distances to determine whether they should swap places or not. If they indeed need to swap, they may initiate a protocol to break their existing set of overlay links and form new links instead.

3) Lowering Algorithm Overhead: Finally, a challenge we face is to lower the overhead of each node running the algorithm on a continuous basis. Each time a node executes steps 3-12 of the algorithm, it needs to (a) initiate a probe, (b) check whether a swap is beneficial, and (c) perform a swap, if beneficial. We have argued that the overhead of step (a) can be made small. Steps (b) and (c) can be expensive, however.

Step (b) requires multiple invocations of the blackbox to measure distance. For some blackbox implementations such as Euclidean vectors, the invocations may be cheap; for others, it may be much more expensive. The cost of step (c) can be even higher, since it involves breaking and forming overlay links, as well as potentially swapping application state. We will discuss step (c) in greater detail in Section V-A.

One way to lower the cost of the algorithm is to reduce the number of probes initiated, both while actively improving the overlay network, and, more importantly, after converging to a point where very little improvement is being achieved. Of course, detecting this global convergence in a distributed way is non-trivial. We would also hope that the number of swaps performed per node is not too large, thus minimizing the cost of step (c).

We can think of the following simple solution: each node continuously monitors the "improvement" in its dissatisfaction index as a result of each of its probes over the last τ minutes. If this improvement is very low, the node realizes that the overlay network has converged to a near-optimal state and can stop initiating probes. This idea does not quite work because each successful swap is only guaranteed to improve the *overall objective function* for the entire network, not an individual node's

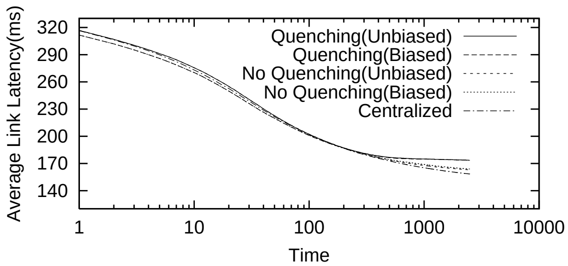

Fig. 6. Convergence of distributed Apocrypha on the Chord network

dissatisfaction index.

A better solution is for each node to monitor its *state fluctuation*, the difference between its best and worst dissatisfaction index over the last τ minutes. A node then continues to "wake up" periodically, but initiates a probe only if its state fluctuation is greater than some threshold ϵ. If the fluctuation is less than ϵ, the node decides to avoid probing, a step we call *probe quenching*. For robustness, a node is also given a small non-zero probability to initiate a probe even if the fluctuation is less than ϵ. So, once the network evolves into a "good" state, the fluctuation is likely to be very low, and the probe initiation rate essentially corresponds to the small probability of initiating a probe for robustness. For reasons that will later become apparent, we will call this probe-quenching mechanism the *hot-stove policy*. Note that nodes always respond to a probe *initiated by some other node*.

B. Evaluation

We now evaluate the use of probes, biasing probes and probe quenching in the optimization of the overlay network. We also investigate the number of swaps required by Apocrypha to optimize the overlay network, and determine its dependence on network size. We present results on the Chord overlay topology operating on top of a GT-ITM model. Results on other overlay topologies and network models are along similar lines and are omitted.

Probes, Biasing and Quenching We consider a 25000-node Chord overlay network starting from an unoptimized configuration. Nodes consider initiating a probe once a minute, and send out the probe if it is not quenched. For the quenching criterion, we use $\tau = 20$ and $\epsilon = 1$ms, i.e., we quench a probe if the dissatisfaction index measured over the last 20 minutes fluctuates by less than 1ms[5].

[5] A point of concern might be that the distance d between the same two nodes might vary over time and that our choice of ϵ is too low. Such cases are handled identically to the way we handle dynamism in the next section.

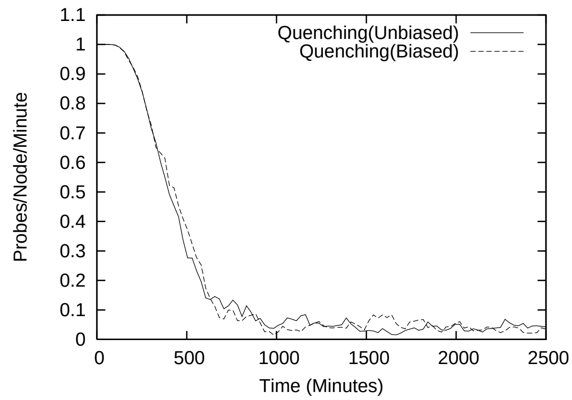

Fig. 7. Number of probes initiated at each time step

Figure 6 depicts the convergence of distributed Apocrypha (using walks with $W = 10$) as a function of time; the centralized algorithm is used as a baseline for comparison. Notice that the different lines mostly coincide, which shows that distributed Apocrypha, with or without biasing and quenching, still performs almost identically to the centralized algorithm. The use of quenching makes the network converge to a slightly inferior state; this is only to be expected, since quenching avoids investing effort in obtaining marginal improvements in overlay quality. (Note that although the graph shows that it takes hundreds of minutes for the system to converge, this convergence time is from an arbitrarily bad state; in practice, the network is maintained incrementally as we discuss in Section V and the convergence time is not relevant.)

Figure 7 shows the average number of probes initiated per node per minute in the network. Note that we always have one probe per node per minute without quenching. We see that the use of probe quenching is effective in reducing the number of probes after very little time. With quenching, each node initiates a probe once every 25 minutes, once the overlay network converges to a reasonable state. We also see that biasing does not appear to either help or hurt in reducing the probes. The real benefit of using biased walks will be seen in our experiments on dynamic overlay networks.

To illustrate the bandwidth cost of performing probes and deciding whether to perform a swap or not, let us assume that the blackbox for measuring distance to a node consists of sending five ping messages and using the median round-trip time to estimate latency. In such a case, the bandwidth cost of performing probes is less than 12 bytes per second, per node, thus being inexpensive. In contrast, in the absence of quenching, the bandwidth cost is 300 bytes/second.

Number of Swaps Figure 8 plots the cumulative number of swaps initiated per node as a function of time. Note

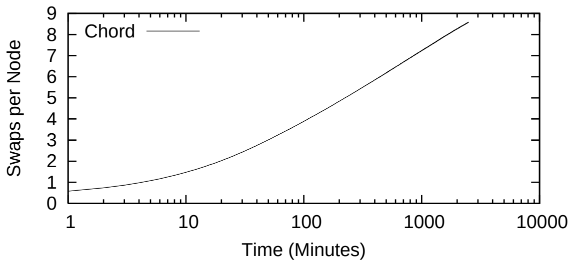

Fig. 8. Number of Swaps performed by Apocrypha

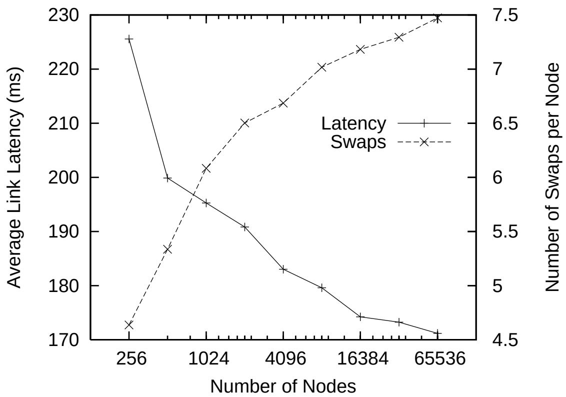

Fig. 9. Quality and Number of Swaps as a function of Network Size

that the x-axis is plotted on a logarithmic scale. We see that the number of swaps increases logarithmically with time, with about half the swaps occurring in the first 100 minutes. Moreover, the total number of swaps performed per node is extremely small, being slightly greater than 8, even after full convergence. We discuss the bandwidth overhead of swaps in Section V-A.

Varying the Number of Nodes We now study how the quality of the overlay network, and the number of swaps performed, depends on the size of the overlay network. Figure 9 depicts both the quality (avg. link latency) of the overlay network, and the total number of swaps performed, for different-sized Chord networks, after 1000 minutes of executing distributed Apocrypha. Note that the x-axis is in logarithmic scale.

We see that the average link latency decreases as the network gets larger. There are two reasons for this phenomenon. First, a smaller Chord network is much more "dense" than a larger network, meaning that a large fraction of all possible edges are present in the network. Consequently, a small network is more constrained and harder to optimize. Second, as the number of nodes increases, there are more pairs of nodes that are close to each other on the physical network, enabling better optimization of overlay links.

Interestingly, the number of swaps performed per node increases roughly *as a logarithmic function of the number of nodes*. (We note that the number of swaps becomes even closer to a logarithmic function if the hill-climbing algorithm is run to "convergence within ϵ" for each network size, rather than for 1000 steps in all cases.) Note that the degree of each node is also close to $\log n$, suggesting that each swap "optimizes" a constant number of links. We have also observed the same "swaps proportional to degree" behaviour on other overlay topologies besides Chord.

V. Dealing with Dynamism

So far, we have ignored the fact that the overlay network changes continuously as nodes join and leave (system *churn*). How well Apocrypha adapts to such dynamism is dependent on how much the overlay topology changes when nodes join and leave. For example, there is not much that the algorithm can do if the entire overlay structure changes every time a node joins or leaves. However, most overlay topologies do not evolve in this fashion; in the topologies we consider, the number of edges that need to be added or removed due to a node arrival or departure is only $O(\log n)$.

Apocrypha adopts the following approach to dealing with dynamic node joins/leaves: The P2P-system protocol is permitted to execute just as it normally does when nodes join and leave. As a result, new overlay links may be created and some old overlay links may be deleted. In consequence, the average overlay link latency may become worse (or better, if we got lucky). However, since a majority of the overlay links do not change, we still expect the overlay network to be in reasonably good shape. Apocrypha can then continue to initiate probes and perform swaps on this new overlay network to improve it, and restore it to its former glory.

Even at high churn rates (with mean node lifetimes under one hour), our experiments show that Apocrypha, with nodes initiating one probe per minute, successfully improves the overlay network to a quality as good as in the static case. The problem is that our hot-stove policy for probe quenching does not help at all in reducing overhead; with a high rate of churn, it is very likely that a node receives at least one new neighbor in the interval of time τ we used as our window. This one new neighbor is likely to throw the state fluctuation over the threshold of ϵ that we set earlier. Consequently, the probe-quenching mechanism hardly ever decides not to perform a probe. Apocrypha continuously performs a lot of work to keep up with all the changes happening in the system in order to improve the system, perhaps more than we want it to. At high rates of churn, we would like the option of performing probes at a lower rate while perhaps settling for a lower-than-optimal-quality overlay network.

Our solution to dealing with churn is based on a slightly counter-intuitive observation. If the overlay network is dynamic, nodes need to be less aggressive in initiating probes. The reason is that we desire to permit some "slack" for the network quality to get slightly worse before we fix it. If no slack is allowed, we will only be wasting a lot of effort making small improvements that will soon disappear if there is more churn. Moreover, by allowing the network to deteriorate slightly, it becomes possible to make larger improvements more efficiently. This leads us to the idea that we should "slow down time" when the network is dynamic, and let nodes react as if in slow motion.

Of course, since we have a distributed system where a node can only observe the dynamism among its neighbors, nodes need to make *local* decisions without any knowledge of the global rate of churn. Despite nodes making such local decisions, we require that the cumulative behaviour of the nodes be a function of the globally observed rate of churn.

We devise a new probe-initiation policy, called the *Pretty-Girl Policy*[6], which provides the above behaviour by using a three-state automaton. Let us define a node to be *young* if it joined the network within the last τ minutes, and old otherwise. Consider a node m in the system, and let k be the number of current neighbors of m that are young. The new probe-initiation policy for a node m now becomes:

- If $k > 0$, node m is in *dynamic mode* and initiates a probe with probability $a + b \cdot k$ for constants a and b.
- If $k = 0$, and m was in dynamic mode earlier, it switches to *tentative-static mode*. Node m slowly increases the probability of initiating a probe until either (a) p probes are sent out in tentative-static mode or (b) k becomes greater than zero. In case (a), the node switches to *static mode*. In case (b), the node switches back to dynamic mode.
- In static mode, a node uses our originally suggested hot-stove policy for quenching.

Some intuition into the above policy can be obtained as follows: It is clear that the aggression of a node in initiating probes should be a function of the dynamism it observes *among its neighbors*, since nodes are required to make local decisions. If very few of a node's neighbors are newly joined, then it ought to be less aggressive and let the newly arrived nodes do the work to improve the system. But if a lot of its neighbors are new, then the node itself needs to step up and initiate probes so that the network does not deteriorate too much.

[6]The basis of this name lies in Albert Einstein's work on understanding external sensory input on time dilation [20].

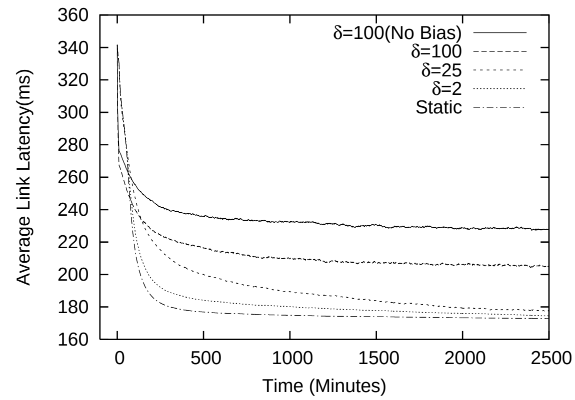

Fig. 10. Convergence as a function of time

The pretty-girl policy helps us deal, not only with different rates of churn, but also with bursts of high churn interspersed with periods of calm. When a burst of churn ends, nodes cautiously send out probes and slowly step up the frequency of probing until they are sure the system is indeed static. Once they realize that it is static, they go back to their usual quenching policy.

Figure 10 depicts the effects of the pretty-girl policy on the convergence of the network in the face of different rates of churn. We again start with an arbitrary, initial Chord overlay network of 25000 nodes and have δ nodes joining and δ nodes leaving the system every minute (for a total of 2δ nodes changing in every minute). Each node considers initiating a probe once a minute and, of course, may not do so if the pretty-girl policy so dictates. We show the evolution of the network for different values of δ. The topmost curve depicts running Apocrypha (at $\delta = 100$) without the use of biased walks, and can be seen to be vastly inferior to using biased walks at the same rate of churn. The remaining curves all use biased walks, and we see that, at low churn rates, the system evolves more or less similarly to the static case. When the churn rate is higher, the network tends to stabilize at a higher average latency rather than climbing all the way down to a quality near that of the static case. The reason is that the pretty-girl policy concludes that it is too expensive to maintain a high-quality network at that high rate of churn.

Figure 11 shows the average number of probes per node per minute at different rates of churn, again as a function of time. The spikes on the left end of the curve may be ignored, as they are artifacts of the system starting with all nodes joining at the same time. We see that the static case performs the least probes, the case $\delta = 2$ performs more probes, and there are even more probes at $\delta = 25$. However, what is interesting is that *fewer* probes are performed at $\delta = 100$ than

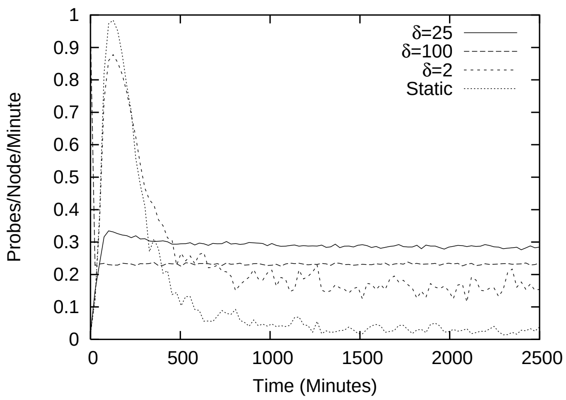

Fig. 11. Number of probes quenched

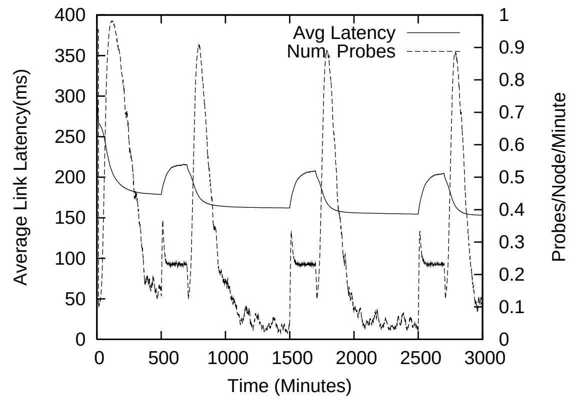

Fig. 12. Effects of burstiness

$\delta = 25$, because Apocrypha realizes that it is not worth performing more probes at that high rate of churn. (Translating these numbers into the bandwidth cost of distance measurements, just as in the earlier section, we see that the probe bandwidth per node using the pretty-girl policy is always less than 90 bytes/second.)

We note that the hot-stove policy would quench practically no probes at all under any rate of churn (e.g., each node performs more than 0.99 probes/minute at $\delta = 100$). At all churn rates, the number of probes initiated per minute using the pretty-girl policy is much lower than that with the hot-stove policy. We also have not shown the evolution of routing *path latency* in the system as a function of time, which is similar to Figure 10.

Figure 12 shows what happens when the churn in the network is bursty. We again start with an initial unoptimized network, and there is no churn except for three bursts in the intervals $[500, 700]$, $[1500, 1700]$ and $[2500, 2700]$. During these intervals, there are $\delta = 100$ nodes joining and $\delta = 100$ nodes leaving the network each minute. The figure shows evolution of both the quality of the network and the number of probes initiated. We see that the number of probes stays relatively low during the bursts of churn, while the quality starts deteriorating. But the deterioration does not get arbitrarily bad (as evidenced by the "flat top" of the humps in the latency curve). When the churn ends, the number of probes drops initially (just before each big spike) while nodes are in tentative-static mode and are sending out "feelers" to check if the network is indeed static. Once nodes detect the absence of churn, they start probing aggressively to improve the network quickly. When the improvement flattens out, they once again reduce the number of probes initiated. Over the long term, we see that the average latency of the network declines steadily.

A. The Cost of Swaps and Using Proxies

One variable we have not shown in our evaluation of dynamic Apocrypha so far is the number of swaps being performed in the system. The number of swaps taking place in the network turns out to be positively correlated to the churn rate δ, which is not surprising. The total number of swaps in the (25000-node) system as a whole is as low as 40 swaps/minute at $\delta = 2$, and going up to 400 swaps/minute at $\delta = 100$.

To understand the costs implied by the above numbers, consider the cost of a swap. A swap requires the breaking of links by each involved node, and the formation of new links instead. This cost, in itself, is fairly small, and can be shown to be well under 4 bytes/second/node. The second source of overhead is the *application state* that may have to be swapped when nodes exchange IDs, whose cost is application-dependent.

In systems using overlays for efficient broadcast, or systems like Gnutella, node IDs are not associated with any application state, making swaps very efficient. Similarly, in P2P multicast systems built on structured overlays [21], swapping is practically free. On the other hand, in P2P distributed hash tables (DHTs), a node's ID determines the content it stores and, therefore, swapping IDs entails having to swap the associated content too.

If the amount of application state is huge, e.g., in a distributed archival system, the swap may be impractical. However, more typical DHT applications have smaller per-node state. For concreteness, we consider an application where each node maintains 1 MB of application state that needs to be swapped. Under this assumption, and at a churn rate of $\delta = 100$ (corresponding to average node lifetimes of about 4 hours), the bandwidth overhead induced by data swaps is 540 bytes/second per node, which is still reasonable.

Using Proxies We can reduce the cost of swaps even further in DHTs, where application state is both small and transient. When two nodes swap IDs, they do not swap their associated content. Instead they set up *proxy links* to each other, so that a query directed to one node for the other's content may be answered by forwarding the query along the proxy link. (Note that new content inserted into the system should be stored at the correct node, and can be reached without the proxy link.) This approach has many advantages.

When a node n joins, the system gives it an ID and requires the node to "take over" a portion of the hash space that some other node m is currently responsible for. Node n can tentatively establish a proxy link to m instead of actually moving content over. Since n has not been in the system very long, it is likely to be highly "dissatisfied" and, consequently, Apocrypha will make n swap IDs multiple times before n "settles down" with some ID. Each time n swaps IDs with some other node n', it can hand over its current proxy link to n', and create a proxy link to n' for itself. When n is no longer involved in swaps for a while, it can move the content over from whichever node its proxy link points to. Thus, the cost of swaps can be reduced considerably.

Second, in many DHT systems, index data stored by a node is designed to "expire" after a certain amount of time. This expiration time is often very low, and the proxy link will quickly become unnecessary as all the data that it points to will soon expire. In the meanwhile, a node can lazily cache all results to queries that are sent over the proxy link, thus improving performance and avoiding copying data that is not queried for. Due to the application-specific nature of this optimization, we do not evaluate its performance in this paper.

VI. Related Work

There have been different approaches adopted for making P2P overlays network-aware. For deterministic topologies, [9] suggested carefully selecting labels in CAN to reflect geographical locations of nodes. However, this approach may be problematic because many properties of CAN rely on the set of labels being randomly chosen. Reference [18] suggests using improved routing protocols specific to CAN; we use the same general principle of exploiting multiple paths in designing Ebola. Reference [5] also suggests a routing scheme with some similarities to Ebola; their routing scheme appears to be for "regular" networks on 2^N nodes.

Pastry [11] and Tapestry [12] use non-deterministic topologies based on hypercubes, and use a strategy analogous to PNS to optimize the network. Both Pastry and Tapestry use smart, practical maintenance algorithms to realize the "optimal configuration" of PNS without having to perform too many distance measurements. Bamboo [22] also uses the same non-deterministic topology, and the PNS approach to optimization, but implements the approach by a hill-climbing strategy to limit optimization costs, a philosophy that is mirrored in Apocrypha. (Note that the hill-climbing in Bamboo is not the same as the hill-climbing performed in Apocrypha.)

There has been theoretical work on constructing specialized overlay networks with guaranteed routing properties, exploiting physical-network information in construction, e.g., [23], [24], [25]. In such cases, the graph structure of the overlay network is itself modified to make it "conform" to physical-network structure, in contrast to our study that focused on pre-defined graph structures. Applying these theoretical constructions in practical settings is a subject of current research.

The idea of using the swap to perform isomorphic transformations is fairly common. Reference [26] uses the swap in order to construct multicast trees. The metric being optimized in this work is different from ours, as it is designed to optimize a tree rather than an arbitrary graph. Moreover, their proposed algorithms are designed for small-scale, static networks, rather than a highly dynamic P2P system. A significant difference between these algorithms and Apocrypha is the quenching policies used by Apocrypha to enable efficient distributed convergence even under dynamic conditions.

VII. Conclusions

We have presented Apocrypha, a generic mechanism to make P2P overlays network-aware. We showed how this mechanism may be applied to deterministic and randomized topologies, leading to optimized structures that could outperform traditional non-deterministic topologies. We explained how Apocrypha operates under dynamic conditions where nodes join and leave the overlay, by trading off the cost of overlay optimization against the utility of optimizing the network in the first place.

References

[1] "Gnutella," Website http://gnutella.wego.com.

[2] M. F. Kaashoek and D. R. Karger, "Koorde: A simple degree-optimal hash table," in *Proc. IPTPS*, 2003.

[3] G. S. Manku, M. Naor, and U. Wieder, "Know thy neighbour's neighbour: The role of lookahead in randomized P2P networks," in *Proc. STOC*, 2004.

[4] M. Castro, P. Druschel, Y. C. Hu, and A. I. T. Rowstron, "Topology-aware routing in structured peer-to-peer overlay networks," in *Proc. Intl. Workshop on Future Directions in Distrib. Computing (FuDiCo 2003)*, 2003, pp. 103–107.

[5] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of DHT routing geometry on resilience and proximity," in *Proc. SIGCOMM*, 2003.

[6] H. Zhang, A. Goel, and R. Govindan, "Incrementally improving lookup latency in distributed hash table systems," in *ACM SIGMETRICS 2003*, June 2003, pp. 114–125.

[7] "Planetlab," Website http://www.planet-lab.org.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. SIGCOMM*, 2001.

[9] S. Ratnasamy, P. Francis, M. Handley, and R. M. Karp, "A scalable Content-Addressable Network," in *Proc. SIGCOMM*, 2001.

[10] P. Fraigniaud and P. Gauron, "An overview of the content-addressable network D2B," in *Proc. PODC*, 2003.

[11] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location, and routing for large-scale peer-to-peer systems," in *Proc. Middleware*, 2001.

[12] 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, Jan. 2004.

[13] G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed hashing in a small world," in *Proc. USITS*, 2003.

[14] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," in *Proc. PODC*, 2002.

[15] G. Pandurangan, P. Raghavan, and E. Upfal, "Building low-diameter P2P networks," in *Proc. FOCS*, 2001.

[16] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. IEEE INFOCOM*, 1996.

[17] "PlanetLab All-Pairs-Pings," http://pdos.lcs.mit.edu/∼strib/pl_app/.

[18] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proc. IEEE INFOCOM-2002*, June 2002.

[19] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris, "Practical, distributed network coordinates," in *Proc. HotNets-II*, 2003.

[20] A. Einstein, "On the effects of external sensory input on time dilation," *Journal of Exothermic Science and Technology*, 1938.

[21] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "SCRIBE: The design of a large-scale event notification infrastructure," in *Proc. NGC*, 2001.

[22] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT," in *Proc. USENIX Annual Technical Conference*, 2004.

[23] I. Abraham, D. Malkhi, and O. Dobzinski, "LAND: Stretch $(1 + \epsilon)$ locality-aware networks for DHTs," in *Proc. SODA*, Jan. 2004, pp. 543–552.

[24] K. Hildrum, J. Kubiatowicz, S. Ma, and S. Rao, "A note on the nearest neighbour in growth-restricted metrics," in *Proc. SODA*, Jan. 2004, pp. 553–554.

[25] D. R. Karger and M. Ruhl, "Finding nearest neighbors in growth-restricted metrics," in *Proc. STOC*, 2002.

[26] S.Banerjee, C.Kommareddy, K.Kar, B.Bhattacharjee, and S.Khuller, "Construction of an efficient overlay multicast infrastructure for real-time applications," in *Proc. INFOCOM*, 2003.

[27] P. Ganesan, K. Gummadi, and H. Garcia-Molina, "Canon in G Major: Designing DHTs with hierarchical structure," in *Proc. 24th Intl. Conf. on Distributed Computing Systems (ICDCS 2004)*, Mar. 2004.

[28] P. Ganesan and G. S. Manku, "Optimal routing in Chord," in *Proc. 15th ACM-SIAM Symp. on Discrete Algorithms (SODA 2004)*, Jan. 2004, pp. 169–178.

[29] Clip2.com, "Clip2 gnutella crawl files," Private collection.

Appendix I
Routing Protocols for Chord

In this section, we describe Chord and different protocols, both well-known and new, that may be used to perform routing on Chord. Recall the definition of Chord: Each node is assigned a unique ID, assumed to be a random integer in $[0, 2^N)$, for a large value of N. For each integer $0 \leq k < N$, a node with id x maintains a link to the node whose ID is the immediate successor of value $x+2^k$, with arithmetic modulo 2^N. The standard means of routing on Chord is to use the greedy routing protocol.

Greedy Routing A message at node x destined for ID y is forwarded along to the neighbor of x that is closest to y without overshooting it [8]. This routing protocol ensures that it is possible to route between any two nodes in an n-node network using only $O(\log n)$ hops with high probability [8]. The expected path length is close to $0.5 \log n$ [27]. Note that greedy routing is oblivious to the *latency* of the different hops, and does not attempt to optimize for the latencies of different hops in any way.

We now introduce three new routing protocols that improve on greedy routing.

A. Choosing paths wisely with Solomon

The Chord structure offers multiple alternative paths from any node to any other node. For simplicity, imagine that there are exactly 2^N nodes with labels $0..2^N - 1$. Greedy routing is then equivalent to expressing the distance to the destination as a binary string, and "fixing" the 1s left-to-right. For example, we would cover a distance of length 56 (111000 in binary) by three successive hops of length 32, 16 and 8, which is equivalent to left-to-right bitfixing on the binary string 111000 to convert the 1s to 0s. We observe that any routing protocol that fixes all these bits will also require the same number of hops, irrespective of the order in which the bits are fixed. The PRS scheme of [5] uses this insight and suggests that all neighbors corresponding to fixing a "1" in the distance be considered *candidates*, and that a node should forward a message to the candidate with the *lowest latency* from itself.

The above idea does not quite work when the number of nodes is not exactly 2^N. For example, let us say we wanted to get from node 0 to node 54 (distance 111000) just as earlier; say we want to take a hop of length 8 to fix the right 1 in the binary string. If there is no node with id exactly 8, the Chord system requires node 0 to maintain a link to the node with the smallest id that is greater than 8. Say, node 0 has a link to node 9. Taking this hop of length 9 would lead to a remaining distance is $56 - 9 = 47$, with a binary representation 101111, which leaves us five more 1s to fix in the distance. In general, the number of hops necessary to reach the destination may not be logarithmic.

One simple way to fix the problem is to consider a neighbor as a candidate for forwarding only if the remaining distance from that neighbor has fewer 1s than the current distance. If no such neighbors are available, we switch to the standard, greedy routing for the remainder of the route. This modified protocol uses at most $2 \log n$ hops to forward a message to a destination. However, such a protocol is too restrictive and does not offer enough choices in selecting paths; so, we relax the condition to also ignore l lower-order bits in the distance. For example, say node 0 needs to route to node 84(distance 1010100). The number of 1s in the distance is 3. If 0 forwards the message to node 17, the remaining distance would be 67(1000011) which has the same number of ones as the original distance. However, if we ignore the last $l = 3$ bits in the distances, the number of ones actually decreases from 2 to 1, and node 17 would be considered a candidate. Ignoring these low-order bits thus increases the number of routing options, and increases the maximum number of hops necessary only by a constant. We call this routing protocol *Solomon* (Selection Of LOcal neighbor from Multiple Overlay Neighbors). The number of bits to ignore, l, should ideally be set to $N - \log n + \log \log n$ where n is the number of nodes in the system and N is the number of bits in the identifiers.

B. Exploiting Bidirectionality: Duomon

Recall that both greedy routing and Solomon use links only in the clockwise(forward) direction. However, many P2P systems implement overlay links as TCP connections, which are inherently bidirectional. In consequence, links can also be used in the anti-clockwise(backward) direction to good effect. For example, if we need to send a message from node 1 to node 0, we could use the insight of Christopher Columbus and simply use the overlay link between 0 and 1 in the anti-clockwise direction. A trivial way to exploit this observation is to trade off bandwidth for latency by simultaneously routing two copies of a query, one clockwise and the other anti-clockwise, using the Solomon protocol. The destination receives two copies of the query, one earlier than the other, and can throw away the duplicate copy. Note that the cost of transmitting the query is also doubled in the process. We call this simple protocol Duomon.

We note that the use of anti-clockwise links is not as symmetric to the use of forward links as it may first appear. The reason is once again, the Chord rule that when a node can't find a neighbor with a specific id i, it looks for a neighbor with an id *larger* than i. A consequence is that the *in-degree* distribution of nodes in Chord, that is, the number of nodes that choose each node as a neighbor, is much more skewed than the out-degree distribution which tends to be about the same for all nodes.

C. Using One-Lookahead: Ebola

Duomon is a simple means of exploiting bidirectional links. However, we can actually do much better, using just one copy of the query, by combining two powerful ideas: the use of both clockwise and anti-clockwise links along the same route, and the use of lookahead.

To illustrate the first idea, imagine we had to cover a distance of $7(111)$. Clockwise routing requires 3 steps of 4, 2 and 1, while anti-clockwise routing would require even more steps on a network with more than 64 nodes. Observe that we could actually cover this distance in *two* steps by combining a forward step of 8 with a backward step of 1. (On a "perfect" Chord network with 2^N nodes, we show that an extension of this idea leads to routing in just $\frac{1}{3}\log n + o(1)$ steps on average in [28].)

The second idea is to use one-lookahead, i.e., make a node aware not only of the quality of links to its neighbors, but also of the links from each neighbor n to n's neighbors. Thus, a node can decide which neighbor to forward a message to, by studying *two steps* of the routing path and evaluating its quality. The advantage of one-lookahead is that nodes can avoid being seduced by attractive-looking neighbors that may not offer a good second step in routing to the destination.

Alternatively, we can think of one-lookahead as an approximation of infinite lookahead; with infinite lookahead, each node can figure out and use the shortest overlay path to route to any destination. As has been observed in [13], the use of one lookahead does not incur much overhead; the lookahead information can be piggybacked on other traffic, and is updated lazily at very low cost when nodes join and leave.

We combine lookahead with bidirectional use of overlay links to produce a routing protocol called *Ebola*(Exploiting Bidirectionality using One LookAhead). Define the *minHamming* distance to the destination as either the clockwise or anti-clockwise distance to the destination, whichever has fewer 1s in its binary representation. The key idea in Ebola is to consider a pair of steps, each of which may be clockwise or anti-clockwise, such that the resulting minHamming distance has fewer 1s than the original minHamming distance. Once all such pairs of steps have been identified, we use the pair that has the lowest total latency[7]. Just as in the case of Solomon routing, if no such pair of steps exists, we revert to greedy routing, along the clockwise or anti-clockwise direction whichever is shorter. Again as in the case of Solomon routing, we will ignore the l least significant bits in distances, while computing minHamming distance.

D. Unidirectional vs. Bidirectional: A Discussion

A point of contention with the use of Ebola is that it may be "unfair" to compare Ebola routing on Chord to unidirectional routing protocols on other topologies, since Ebola is effectively routing on a network with twice as many links. Our approach is to consider all topologies as being inherently undirected (and, therefore, treat all links as bidirectional). For some topologies like Chord, the bidirectionality of links helps improve the number of routing hops necessary. For other topologies such as de Bruijn graphs, the bidirectionality may not help as much. All inter-topology comparisons in this paper ensure that the average degree of nodes, *treating the graphs as undirected*, are nearly the same in all the topologies.

A second argument that may be raised with Ebola is that its use of lookahead lets it "cheat" by increasing each node's effective degree. However, the use of lookahead is *not* equivalent to working with a higher-degree network as far as maintenance costs go. It is a lot cheaper to maintain look-ahead information at low cost, as discussed in [13].

E. Evaluation

We now evaluate the improvement obtained in query latency by using each of these three new routing protocols on a static Chord network as Apocrypha optimizes it.

1) Evaluation on GT-ITM: We begin by considering a static 25,000-node Chord network with the physical network modeled using GT-ITM. Figure 13 plots the average path latency between all pairs of nodes for the three new routing protocols, as well for greedy clockwise routing. Note that we have normalized the average path latency on the y-axis by the average latency between nodes on the physical network to obtain the *stretch*. So, a stretch of 2 implies that it takes twice as long, on average, to route from one node to another on the overlay network, compared to just directly routing between the nodes on the physical network.

[7] We eliminate pairs of steps from node i that actually leads to a direct neighbor of i.

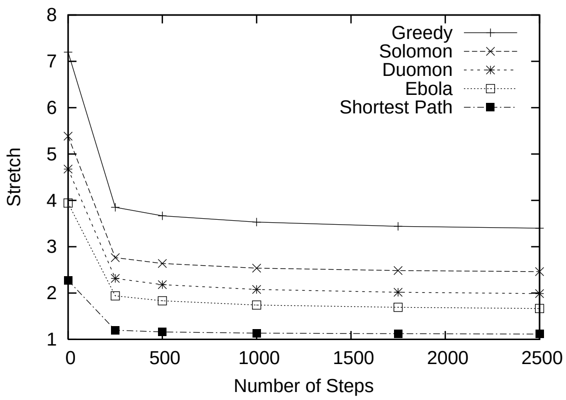

Fig. 13. Average Path Latency on Chord with different routing protocols (GT-ITM)

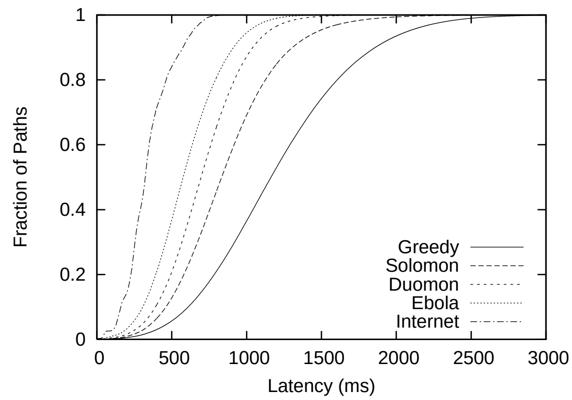

Fig. 14. Cdf of query latencies using different routing protocols

Note that the initial stretch, before Apocrypha performs any optimizations, is 7 for the standard greedy protocol (which is not too surprising, since greedy routing takes about 7 overlay hops on average, and each overlay hop should initially correspond to a random hop on the internet). On the other hand, Ebola has an initial stretch of under 4, Duomon has a stretch 4.7 and Solomon 5.3, thus demonstrating their utility even when Apocrypha is not used to optimize the Chord network. When used in conjunction with Apocrypha, all of our routing protocols continue to perform considerably better than greedy routing. For example, the use of Ebola reduces the stretch to under 1.7 on the static Chord network.

Figure 14 shows the cumulative distributions of the path latencies when using the different routing protocols, after the network has been optimized by Apocrypha for 1000 steps (which corresponds to the quality of the network seen under typical rates of dynamism (as seen in Figure 10). The left-most curve represents the distribution of the latencies between pairs of nodes in the GT-ITM model. For greedy routing (bottom-most curve), we see that the percentage of paths with a latency under 1 second is only about 30%. On the other hand, for Ebola, more than 93% of the paths have a latency under 1 second. Greedy routing has a much higher variance than our three protocols, which is not surprising since greedy routing is deterministic and independent of the individual link characteristics.

2) Evaluation on PlanetLab: We now evaluate the routing protocols on Chord, this time on the PlanetLab data. As earlier, we construct a 182-node Chord network, with inter-node latencies being determined by ping times on PlanetLab. Figure 15 depicts the stretch for the different routing protocols as a function of the number of hill-climbing steps.

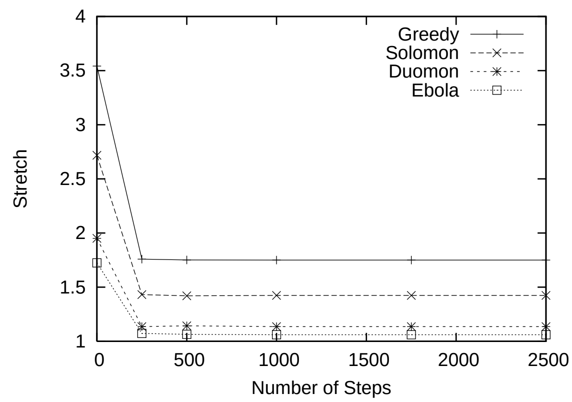

Fig. 15. Average Path Latency on Chord with different routing protocols

Greedy routing, as depicted in the top line, has a stretch of 3.5 on the unoptimized Chord network, which is the average number of hoprs required for routing on a 182-node Chord network. After 250 steps, the stretch drops to 1.75 and remains constant at that value. The use of Solomon reduces the initial stretch to 2.7, and the stretch on the optimized network falls below 1.5. With the use of Ebola, the initial stretch is 1.7, and drops to less than 1.06 after optimization, which is very close to the optimal achievable stretch.

Overall, we observe that the stretch incurred by the routing protocols is lower than in the experiments on the GT-ITM topology, again suggesting that it is more difficult to optimize a network in the GT-ITM model than on PlanetLab. We conjecture that the underlying reason for this phenomenon is that there are more nodes that are "close" to a given node on PlanetLab than in the GT-ITM model.

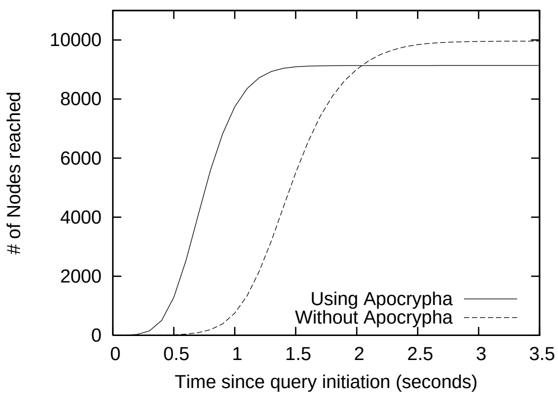

Fig. 16. Number of nodes reached in Gnutella as a function of time.

APPENDIX II
APPLYING APOCRYPHA TO GNUTELLA

In this section, we study the impact of Apocrypha on queries in the Gnutella network. Gnutella uses flooding with a TTL in order to find answers for a query. The metric of concern in Gnutella is the number of answers obtained for a query as a function of the time taken to obtain that many answers. Since different queries may have different numbers of answers available, and the answers may be distributed across nodes in different ways, we use a slightly simpler metric: the number of nodes a query is propagated to, as a function of the time since the query was initiated. Ideally, we would like the query to propagate to the most nodes in the least amount of time.

We simulated the Gnutella network on top of our GT-ITM network model, using a Gnutella topology obtained by a crawl of the real Gnutella network [29]. Since we expected bandwidth to be an issue in a flooding-based system such as Gnutella, we imposed bandwidth constraints on the physical tail-links connecting each of the nodes to a router in our network model. Low-degree nodes were assumed to be attached to the network via cable modems, higher-degree nodes through 10Mbps LANs and the highest-degree nodes via 100Mbps LANs. We also introduced limited buffering on the tail links, and each node dropped network packets when the buffer overflowed. In our simulation, each node generated queries by a Poisson process with a rate of one query/minute. Queries were flooded using an initial TTL of 5. The CPU time taken to process a query is considered negligible compared to the time taken for transmitting messages over the overlay network.

We simulated both an unoptimized Gnutella network with nodes randomly located on the physical network, and a Gnutella network optimized by Apocrypha. Since the Gnutella network converged very quickly with the use of Apocrypha, we show only one representative state

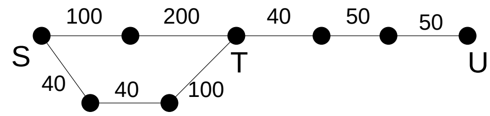

Fig. 17. An example of a long circuit

of the Gnutella network rather than showing it at various stages of convergence.

Figure 16 shows the cumulative number of nodes reached by an individual query as a function of time. We see that, initially, the query reaches more nodes in less time on the optimized network compared to the unoptimized one. After one second, the query has reached about 9000 nodes on the optimized network comparted to fewer than 1000 nodes in the unoptimized case. Eventually, however, the use of Apocrypha *reduces* the total number of nodes reached (9000 with its use versus 10000 otherwise). One may initially believe that the optimized network may result in more load imbalance, hence causing more nodes to become bandwidth bottlenecks. Upon closer inspection, we found that the real reason for reaching fewer nodes is due to a phenomenon that we call *long circuits.* We now illustrate this phenomenon.

A. *The Long-circuit phenomenon*

The Gnutella protocol uses a TTL field to limit the broadcast of a query to a certain depth h. One would assume that all nodes within h hops would receive the query if there were no bottlenecks. Unfortunately, due to the way nodes throw away duplicate query messages, not all nodes within h hops may receive the query. We illustrate this problem in Figure 17, which shows a Gnutella-network fragment with numbers on the edges representing latencies. Say a query is initiated from node S with a TTL of 5. We observe that node T has two paths from S, one of length two and the other of length three. If the query from S reaches T via the 3-hop path first, as it does here, the query would be forwarded from T with a TTL of 2, which means that node U will not receive a copy of the query. When a duplicate copy of the query is received via the 2-hop path, it would simply be discarded. Thus, the query fails to reach U, despite U being within five hops of the source S. When there are two paths between S and T with the longer path being faster, we say S and T have a *long circuit.*

Long circuits occur naturally in Gnutella, even without the application of Apocrypha. However, the use of Apocrypha exacerbates the problem, because of its side-effect of increasing the variance in link latencies for a given node. After Apocrypha optimizes the overlay network, a typical node has five or six fast links and one or two slow links. Following two of these slow links may lead to a node that has a faster alternative route with more hops, thus creating a long circuit. In consequence, fewer nodes end up being reached by the query. The increase in the long-circuit phenomenon is the primary culprit in causing queries on the optimized network to reach fewer nodes.

There are different solutions possible for eliminating long circuits. We could (a) alter the protocol to retransmit a duplicate copy of the query on detecting a long ciruit, (b) *modify* the topology to eliminate long circuits, (c) alter Apocrypha to optimize a different objective function: the sum of the maximum latency links for each node, or (d) use new ad hoc topologies in combination with new search protocols to avoid long circuits.

B. Other Ad Hoc Networks and Extensions

Supernode networks are very similar to Gnutella networks except that all the supernodes tend to have high bandwidth and more homogeneous “capacities”. So, we expect supernode networks to be more like random regular graphs than power-law graphs. Apocrypha is equally applicable to such networks. An alternative to Gnutella flooding is random-walk based searches on random regular graphs. In this case, the application metric of minimizing query latency is exactly the same as our hill-climbing metric of minimizing the average overlay-link latency.

So far, we have assumed that all nodes have the same capabilities and that any pair of nodes can swap their set of neighbors, irrespective of the relative sizes of these neighbor sets. In reality, some nodes are capable of maintaining connections with a large set of neighbors, while others operating via a cable modem or a DSL line might not be able to support too many neighbors. It is easy to modify Apocrypha to take these simple *capacity constraints* into account. Recall that distributed Apocrypha requires two nodes to agree on whether to swap their labels or not. We could simply let a node refuse to swap labels if doing so leads to it having an unreasonably high degree. We omit details.