The EGOIST Overlay Routing System

Computer Science department, Boston University
Tech. Rep. BUCS-TR-2008-016
July 1, 2008

Georgios Smaragdakis
Boston University
gsmaragd@cs.bu.edu

Vassilis Lekakis
FORTH & University of Crete
lekakis@ics.forth.gr

Nikolaos Laoutaris
Telefónica Research, Barcelona
nikos@tid.es

Azer Bestavros
Boston University
best@cs.bu.edu

John W. Byers
Boston University
byers@cs.bu.edu

Mema Roussopoulos
Institute of Computer Science, FORTH
mema@ics.forth.gr

A foundational issue underlying many overlay network applications ranging from routing to peer-to-peer file sharing is that of connectivity management, *i.e.*, folding new arrivals into an existing overlay, and rewiring to cope with changing network conditions. Previous work has considered the problem from two perspectives: devising practical heuristics for specific applications designed to work well in real deployments, and providing abstractions for the underlying problem that are analytically tractable, especially via game-theoretic analysis. In this paper, we unify these two thrusts by using insights gleaned from novel, realistic theoretic models in the design of Egoist – a distributed overlay routing system that we implemented, deployed, and evaluated on PlanetLab. Using extensive measurements of paths between nodes, we demonstrate that Egoist's neighbor selection primitives significantly outperform existing heuristics on a variety of performance metrics, including delay, available bandwidth, and node utilization. Moreover, we demonstrate that Egoist is competitive with an optimal, but unscalable full-mesh approach, remains highly effective under significant churn, is robust to cheating, and incurs minimal overhead. Finally, we use a multiplayer peer-to-peer game to demonstrate the value of Egoist to end-user applications. This technical report supersedes BUCS-TR-2007-013.

1. INTRODUCTION

Motivation: Overlay networks are used for a variety of applications including routing [1], content distribution [11, 38], peer-to-peer file sharing, data-center applications [18], and online multiple multiplayer games [5]. A foundational issue underlying many such overlay network applications is that of connectivity management. Connectivity management is called upon when having to wire a newcomer into the existing mesh of nodes (bootstrapping), or when having to rewire the links between overlay nodes to deal with churn and changing network conditions. Connectivity management is particularly challenging for overlay networks because overlays often consist of nodes that are distributed across multiple administrative domains, in which auditing or enforcing global behavior can be difficult or impossible. As such, these nodes may act selfishly to maximize the benefit they receive from the network, as exemplified in studies relating to selfish (source) routing [26] and free riding [13] in P2P file-sharing networks.

Selfish behavior has many implications for connectivity management. In particular, it creates additional incentives for nodes to rewire, not only for operational purposes (bootstrapping and substituting nodes that went off-line), but also for seizing opportunities to incrementally maximize the local connection quality to the overlay. While much attention has been paid to the harmful downsides of selfish behavior, the impact of adopting selfish connectivity management techniques in *real* overlay networks has received very little attention. In our work, we dwell not on the negatives, but instead focus on the potential benefits from such selfish behavior, which include the obvious benefits to selfish nodes, but more surprisingly, to the network as a whole. Indeed, we confirm that selfishness is not the problem, so much as inaction, indifference, or naive reaction: all of which incur high social costs. Our paper addresses these issues by providing a methodical evaluation of the design space for connectivity management in overlay

This work is supported in part by a number of NSF awards, including CISE/CSR Award #0720604, ENG/EFRI Award #0735974, CISE/CNS Award #0524477, CISE/CNS Award #0520166, CNS/ITR Award #0205294, CISE/EIA RI Award #0202067, CAREER Grant #0446522, and the European Commission (Contract No. RIDS-011923).

networks, including the demonstration of the implications and promise from adopting a selfish approach to neighbor selection in real network overlays.

Selfish Neighbor Selection: In a typical overlay network, a node must select a fixed number (k) of immediate overlay neighbors for routing traffic or queries for files.[1] Previous work has considered this problem from two perspectives: (1) devising *practical heuristics* for specific applications in real deployments, such as bootstrapping by choosing the k closest links, or by choosing k random links in a P2P file-sharing system; and (2) providing abstractions of the underlying fundamental neighbor selection problem, which are amenable to theoretical formulation and analysis as exemplified in the recent work on Selfish Neighbor Selection (SNS) [19, 17]. This SNS formulation focused on characterizing the emergent overlay topology when overlay nodes behave selfishly and employ "Best-Response" (BR) neighbor selection strategies. Using BR a node chooses the best k neighbors that optimize its connection quality to the overlay, granted knowledge of how other nodes have connected among themselves.

This prior work demonstrates that selfish players can select neighbors so as to efficiently reach near-equilibria in the Nash sense, while also providing good global performance. One implication from that prior work is that shortest-path overlay routing performs much better over SNS topologies than over random and myopic ones. Left unanswered in this prior work, though, is whether it is practical to build SNS-inspired overlays, how to incorporate additional metrics other than delay, *e.g.*, bandwidth, whether such overlays would be robust against network dynamics and whether they would scale.

Paper Scope and Contributions: In this paper we address the questions mentioned above and describe the design, implementation, and evaluation of EGOIST: an SNS-inspired prototype *overlay routing network* for PlanetLab. EGOIST serves as a building block for the construction of efficient and scalable overlay applications consisting of (potentially) selfish nodes.

Our contributions can be summarized as follows. We first demonstrate through real measurements on PlanetLab that overlay routing atop EGOIST is significantly more efficient than systems utilizing common heuristic neighbor selection strategies under multiple performance metrics, including delay, system load and available bandwidth. Second, we demonstrate that the performance of EGOIST approaches that of a (theoretically-optimal) full-mesh topology, while achieving superior scalability, requiring link announcements proportional to nk compared to n^2 for a full mesh topology. We also demonstrate that the computational, memory and traffic overhead to create and operate EGOIST in minimal. Third, to accommodate high-churn environments, we introduce a hybrid extension of the "Best-Response" (BR) neighbor selection strategy, in which nodes "donate" a portion of their k links to the system to assure connectivity, leaving the remaining links to be chosen selfishly by the node. Our experiments show that such an extension is warranted, especially when the churn rate is high relative to the size of the network. Fourth, we consider the impact of cheaters – nodes that announce false information in order to benefit themselves, or harm the network. While such behavior can be identified and eliminated through the use of appropriate mechanisms, we show that EGOIST remains robust even without the use of such mechanisms. Finally, we discuss how EGOIST can provide a redirection stepping-stone for the benefit of end-user applications including file transfer and multiplayer peer-to-peer games. The list of EGOIST artifacts is provided in Section 6.

2. BACKGROUND

2.1 Basic Definitions

Let $V = \{v_1, v_2, \ldots, v_n\}$ denote a set of overlay routing nodes. Node v_i establishes a *wiring* $s_i = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ by creating links to k other nodes (we will use the terms link, wire, and edge interchangeably). Edges are *directed* and *weighted*, thus $e = (v_i, v_j)$ can only be crossed in the direction from v_i to v_j, and has cost d_{ij} (in general, $d_{ji} \neq d_{ij}$). Let $S = \{s_1, s_2, \ldots, s_n\}$ denote a *global wiring* between the nodes of V and let $d_S(v_i, v_j)$ denote the cost of a shortest directed path[2] between v_i and v_j over this global wiring; $d_S(v_i, v_j) = M \gg \max_{i,j} d_{ij}$, if there is no directed path connecting the two nodes. For the overlay networks discussed here, the above definition of cost amounts to the incurred end-to-end delay when performing shortest-path routing along the overlay topology S, whose direct links have weights that capture the delay of the underlying IP path connecting one end of the overlay link to the other. Let $C_i(S)$ denote the cost of v_i under the global wiring S, defined as a weighted summation of its distances to all other nodes, *i.e.*, $C_i(S) = \sum_{j=1, j\neq i}^{n} p_{ij} \cdot d_S(v_i, v_j)$, where the weight p_{ij} denotes "preference" *e.g.*, the percentage of v_i's traffic that is destined to node v_j.

DEFINITION 1. *Best-Response (BR) Given a residual wiring $S_{-i} = S - \{s_i\}$, a best response for node v_i is a wiring $s_i \in S_i$ such that $C_i(S_{-i}+\{s_i\}) \leq C_i(S_{-i}+\{s'_i\})$, $\forall s'_i \neq s_i$, where S_i is the set of all possible wirings for v_i.*

[1] Hard constraints on the number of first hop neighbors are imposed in most peer-to-peer systems to address scalability issues, up-link and down-link fragmentation, and CPU consumption due to contention.

[2] In our implementation, we computed shortest path using Dijkstra's algorithm. Given than the graph is sparse, we used the most efficient implementation of the algorithm using Fibonacci heap that requires $O(|E| + |V| \log |V|)$ amortized time, where $|E|$ is the number of edges in the graph.

The *Selfish Neighbor Selection* (SNS) game was introduced in [19] as a strategic game where nodes are the players, wirings are the strategies, and C_i's are the cost functions. It was shown that under hop-count distance, obtaining the BR of v_i requires solving an asymmetric k-median problem on the residual wiring S_{-i} and is, therefore, NP-hard. To overcome the computational obstacle, we applied the local search heuristic [2] that provides a solution in a polynomial number of iterations. Experimental results showed that the performance of the above heuristic is within 5% of the optimal [19].

2.2 Related Work

Our work is inspired by the SNS game [19, 17]. While these works presented basic theoretic and experimental results, they did not consider any of the practical systems issues that are covered in this paper, such as dealing with churn in realistic network conditions or achieving high global performance without the computational and control message overheads required by theoretical formulations. Network Creation Games that predate SNS [12, 9, 25, 8] have considered settings in which nodes may buy as many links (neighbors) as they like and thus differ fundamentally from our work, in which constraints on the number of neighbors play a central role.Also, fundamentally different is the work on Selfish Routing [26, 30], in which the network topology is part of the input to the game, and selfish source routing is the outcome. In a way, this is the inverse of our work, in which network-based (shortest-path) routing is an input of the game, and topology is the outcome. Selfish Routing is also based on source routing which is either not provided in most system implementations, or it is difficult to perform well in systems with high churn like peer-to-peer systems.

A number of routing overlay systems have been recently proposed [31, 1, 22, 21, 40, 23, 16, 41, 32, 36]. Most of these have been proposed as ways of coping with some of the inefficiencies of native IP routing. The basic design pattern is more or less the same: overlay nodes monitor the characteristics of the overlay links between them (overlay topology may differ among systems) and employ a full-fledged or simpler [16] routing protocol to route at the overlay layer. Some overlay routing systems optimize route hop count [21, 32, 36], others optimize for application delay [31, 1, 26, 22, 40, 23, 16], and others optimize for available bandwidth [41]. These works assume that either all overlay nodes are under central control and thus obediently follow simple empirical neighbor selection strategies as discussed earlier, or bypass the issue altogether by assuming that some fixed overlay design is already in place. With reference to the employed metric, in our work, we provide mechanisms to support optimization of *all* aforementioned metrics and leave it up to the application designers to choose the most suitable one.

In structured DHTs, proximity neighbor selection has been proposed to make the overlay topology match the underlying IP topology as much as possible [27, 15] in order to achieve faster lookups: Nodes can choose the physically closest nodes from a set of candidate nodes. While this approach gives to nodes some flexibility in choosing neighbors selfishly, the set of nodes from which the choice can be made is constrained by node ID and thus tuning it at will becomes impossible [37]. Undoubtedly, DHTs are able to provide the best possible indexing of objects in a network. On the other hand, routing of traffic on DHTs has been shown to be sub-optimal due to local forwarding [17, 24]. EGOIST can be integrated as a different layer in DHTs; when an object is mapped onto a node, EGOIST is responsible to optimally route the content.

3. THE EGOIST OVERLAY SYSTEM

In this section we overview the basic design of our EGOIST overlay routing system.

3.1 Basic Design of EGOIST

EGOIST is a distributed system that allows the creation and maintenance of an overlay network (evaluated on PlanetLab), in which every node selects and continuously updates its k overlay neighbors in a selfish manner—namely to minimize its (weighted) sum of distances to all destinations under shortest-path routing. For ease of presentation, we will assume that *delay* is used to reflect the cost of a path, noting that other metrics – which we will discuss later in the paper and which are incorporated in EGOIST's implementation – could well be used to account for cost, including bandwidth and node utilization.

In EGOIST, a *newcomer* overlay node v_i connects to the system by querying a *bootstrap* node, from which it receives a list of *potential* overlay neighbors. The newcomer connects to at least one of these nodes, enabling it to participate in the link-state routing protocol running at the overlay layer. As a result, after some time, v_i obtains the full residual graph G_{-i} of the overlay. By running all-pairs shortest path algorithm on G_{-i}, the newcomer is able to obtain the pair-wise distance (delay) function $d_{G_{-i}}$. In addition to this information, the newcomer estimates d_{ij}, the weight of a potential direct overlay link from itself to node v_j, for all $v_j \in V_{-i}$. Using the values of d_{ij} and $d_{G_{-i}}$, the newcomer connects to G_{-i} using one of a number of wiring policies (discussed in Section 3.2). In our implementation, each node acts as a server that listens to all the messages of the link state protocol and propagates them only to its immediate neighbors. In order to reduce the traffic in the system, each node propagates only unique messages by dropping messages that have been received more than once or have been superseded. There are also two threads, one for estimating d_{ij}, and one responsible for estimating the new wiring and propagating the wiring to the immediate neighbors. In order to minimize the load in the system, a node propagates its wiring to its immediate neighbors only if this changes.

Clearly, obtaining d_{ij} for all n nodes requires $O(n^2)$ measurements.[3] However, we note that these $O(n^2)$ measurements do not have to be announced or be continuously monitored. In particular, each node needs to monitor and send updates only for the k links that it chooses to establish, with $O(n)$ measurements to all nodes in the overlay done much less frequently – namely once per *wiring epoch*, which is defined as the period T between two successive evaluations by a node of its set of candidate links and possible adoption of a new wiring (*i.e.*, re-wiring) based on such evaluation. Since re-wiring is much less frequent than monitoring of the established k links, the load imposed by the link-state protocol is only $O(nk)$ and not $O(n^2)$.

3.2 Neighbor Selection Policies in EGOIST

As its namesake suggests, the default neighbor selection policy in EGOIST is the Best-Response (BR) strategy described in Section 2.1, and detailed in [19]. Using BR, a node selects all its k neighbors so as to minimize a local cost function, which could be expressed in terms of some performance metric (*e.g.*, average delay to all destinations, maximum aggregate throughput to all destinations, etc). Since obtaining an exact BR is computational expensive under both delay [19] and throughput, in Section 4.1, we employ fast approximate versions based on local search (that was introduced in Section 2.1) to reduce computational costs and enhance scalability. In addition to BR, we have also implemented the following neighbor selection policies in order to perform a comparative evaluation.

***k*-Random**: Each node selects k neighbors randomly. If the resulting graph is not connected, we re-wire some links to enforce a cycle upon it.

***k*-Closest**: Each node selects its k neighbors to be the nodes with the minimum link cost (*e.g.*, minimum delay from it, maximum bandwidth, *etc.*). Again, if the graph is not connected, we enforce a cycle.

***k*-Regular**: In this case, all nodes follow the same wiring pattern dictated by a common offset vector $o = \{o_1, o_2, \ldots, o_k\}$, used as follows: node i connects to nodes $i + o_j \mod n$, $j = 1, \ldots, k$. In our system, we set $o_j = 1 + (j-1) \cdot \frac{n-1}{k+1}$.One way to visualize this is to consider that all nodes are placed on a ring according to their ids (as with a DHT). Thus, an offset vector makes each node use its k links to connect to other nodes so as to equally divide the periphery of the ring.

[3] Notice that d_{ij} can be obtained through active or passive measurements depending on the metric of interest (see Section 4.1 for details).

3.3 Dealing with Churn in EGOIST

EGOIST's BR neighbor selection strategy assumes that existing nodes never leave the overlay. Therefore, even in an extreme case in which some nodes are reachable through only a unique path, a node can count on this path always being in place (re-wirings by other nodes will not tear it down as this would also disconnect them [17]). Overlay routing networks (*e.g.*, RON [1]) are not inherently prone to churn to the extent that file-sharing P2P-networks [14, 28] are. Nonetheless, nodes may occasionally go down, or network problems may cause transient disconnections until successive re-wirings establish new paths. One could re-formulate the BR objective function used by a node to take into account the churning behavior of other nodes. This, however, requires modeling of the churn characteristics of various nodes in an overlay, which may not be feasible, particularly for large networks [39].

In EGOIST we follow a different approach reminiscent of how k-Random and k-Closest policies ensure overlay connectivity. We introduce a hybrid wiring strategy (HybridBR), in which each node uses k_1 of its k links to selfishly optimize its performance using BR, and "donates" the remaining $k_2 = k - k_1$ links to the system to be used for assuring basic connectivity under churn. We call this wiring "hybrid" because, in effect, two wiring strategies are in play – a selfish BR strategy that aims to maximize local performance and a selfless strategy that aims to maintain global connectivity by providing redundant routes.

There are several ways in which a system can use the k_2 donated links of each node to build a connectivity backbone. Young et al. [40] proposed the use of k Minimum Spanning Trees (k-MST). Using k-MST (a centralized construction) to maintain connectivity is problematic, as it must always be updated (due to churn and to changes in edge weights over time), not to mention the overhead and complexities involved in establishing $(k_2/2)$-MSTs. To avoid these complexities, EGOIST uses a simpler solution that forms $k_2/2$ bidirectional cycles. Consider the simplest case $k_2 = 2$, which allows for the creation of a single bidirectional cycle. To accommodate a new node v_{n+1}, node v_n will disconnect from node v_1 and connect to v_{n+1}, whereas the latter will connect to v_1 to close the cycle. For higher $k_2/2$, the system decides $k_2/2$ *offsets* and then each node connects to the nodes taken by adding (modulo n) its id to each offset. If k_2 is small (*e.g.*, 2) then the nodes will need to monitor (*e.g.*, ping) the backbone links closely so as to quickly identify and restore disconnections. With higher k_2 the monitoring can be more relaxed due to the existence of alternative routes through other cycles. Computing BR using k_1 links *granted* the existence of the k_2 links can be achieved by restricting the set candidate candidate immediate neighbors for swapping.

We have implemented HybridBR in Egoist. As hinted above, donated links are monitored aggressively so as to recover promptly from any disconnections in the connectivity backbone through the use of frequent heartbeat signaling. On the other hand, the monitoring and upkeep of the remaining BR links could be done lazily, namely by measuring link costs, and recomputing BR wirings at a pace that is convenient to the node—a pace that reduces probing and computational overheads without risking global connectivity.

To differentiate between these two types of link monitoring strategies (aggressive versus lazy), in Egoist we allow re-wiring of a dropped link to be performed in one of two different modes: *immediate* and *delayed*. In immediate mode, re-wiring is done as soon as it is determined that the link is dropped, whereas in delayed mode re-wiring is only performed (if necessary) at the preset *wiring epoch* T. Unless otherwise specified, we assume a delayed re-wiring mode is in use.

3.4 Dealing with Cheaters in Egoist

In this paper the selfishness in the selection of neighbors has the game theoretic meaning of local optimization and does not imply any anti-social behavior that needs to be mitigated. In this section, we briefly examine such harmful ways in which a node may "cheat" its way through, as well as possible countermeasures.

The most blatant form of cheating is *free-riding*, *i.e.*, using the system to route one's own traffic but denying routing to any incoming traffic from other nodes. Dealing with such behavior has been the subject of a number of studies, including the works in [6, 7] which propose the adoption of reputation and repudiation or punishment mechanisms that act as incentives for nodes to route, and/or expel misbehaving nodes from the system. These studies are orthogonal to and thus complement our work.

A more elaborate way for a node to cheat is to announce false information via the link-state protocol to discourage others from picking it as an upstream neighbor. For example, a node can cheat by falsely announcing larger-than-actual delays for its potential outgoing links. One could add mechanisms to detect this type of cheating. If the construction of the overlay is based on passive measurements obtained from a virtual coordinate system (as discussed in Section 4.1), then nodes could periodically select a random subset of remote nodes and "audit them" by querying the coordinate system for the delays of the outgoing links of the audited nodes and comparing them to the actual values that the audited nodes declare on the link-state routing protocol. Similar audits can be designed using active probing by sending traffic and measuring its delay and comparing it to the expected delay based on the delays that nodes on the end-to-end path declare. In Section 4.5, we evaluate the impact of broadcasting false information to cheat the system: we show that even without the use of the aforementioned audit mechanisms, Egoist is robust to this form of cheating.

4. EXPERIMENTAL EVALUATION

4.1 Cost Metrics

As alluded earlier, a number of metrics can be used to measure the "cost" of traversing an overlay link. Clearly, the choice of an appropriate one depends largely on the application at hand. In this section, we review the various metrics we have incorporated in Egoist .

Link and Path Delays: Delays are natural cost metrics for many applications, especially those involving interactive communication. To obtain the delay cost metric, a node needs to obtain estimates for its own delay to potential neighbors, and for the delay between pairs of overlay nodes already in the network. In Egoist , we estimate directed (one-way) link delays using two different methods: an active method based on `ping`, and a passive method using the `pyxida` virtual coordinate system [20]. Using `ping`, one-way delay is estimated to be one half of the measured `ping` round-trip-times (RTT) averaged over enough samples. Clearly, a node is able to measure such a value for all of its direct (overlay) neighbors, and is also able to relay such information to any other nodes through the overlay link-state routing protocol. To estimate the distance to nodes that were configured not to reply to `ping`, we used application layer `ping`. Using `pyxida`, delay estimates are available through a simple query to the `pyxida` system.

Node Load: For many overlay applications, it may be the case that the primary determinant of the cost of a path is the performance of the nodes along that path—*e.g.*, if traversal of nodes along the path incur significant overhead due to (say) context switching and frequent crossing of user/kernel spaces. Thus, in Egoist , we allow the use of a variation of the delay metric in which all outgoing links from a node are assigned the same cost, which is set to be equal to the measured load of the node. When applicable, the estimation of such a metric is straightforward as it requires only local measurements. In Egoist , we did this by querying the CPU load of the local PlanetLab node, and computing an exponentially-weighted moving average of that load calculated over a given interval (taken to be 1 minute in our experiments querying the `loadavg` reports).

Available Bandwidth: Another important cost metric, especially for content-delivery applications, is the available bandwidth on overlay links. Different available bandwidth estimation tools have been proposed in

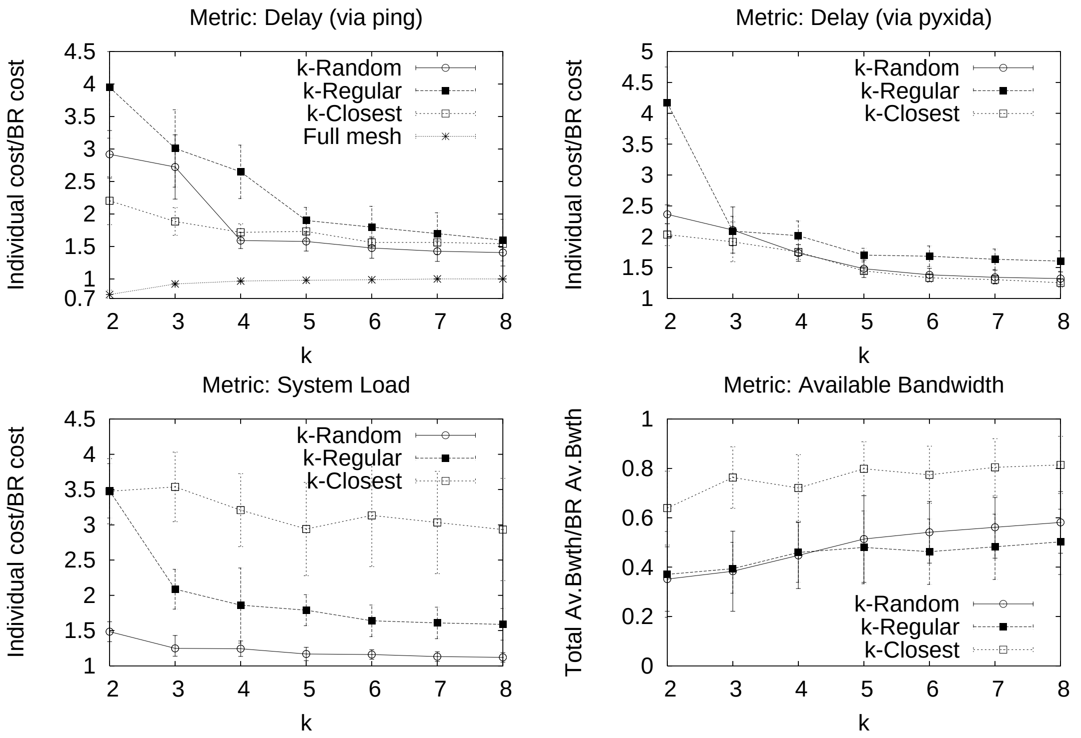

Figure 1: PlanetLab baseline experiments showing the individual costs for various neighbor selection policies (normalized with respect to BR costs) as a function of number of neighbors k for a 50-node EGOIST overlay: Cost metric is `ping` delays (top-left), `pyxida` delays (top-right), node CPU load (bottom-left), and available bandwidth (bottom-right).

the literature [33]. In EGOIST , we used `pathChirp` [29], a light-weight, fast and accurate tool, which fits well with PlanetLab-specific constraints, namely: it does not impose a high load on PlanetLab nodes since it does not require the transmission of long sequences of packet trains, and it does not exceed the max-burst limits of Planetlab. `pathChirp` is an end-to-end active probing tool, which requires the installation of sender and receiver `pathChirp` functionality in each EGOIST node. The available bandwidth between a pair of nodes $v, u \in V_{-i}$ is given by:

$$AvailBW(v,u) = \max_{p \in P(v,u)} AvailBW(p),$$

where the available bandwidth for a path p is given by:

$$AvailBW(p) = \min_{e \in p} AvailBW(e),$$

and $P(v,u)$ denotes the set of paths that connects v to u. Thus, finding $P^*(v,u)$ that maximizes the available bandwidth between v and u, and the bottleneck edge, is a "Maximum Bottleneck Bandwidth" problem which can be solved using a simple modification of Dijkstra's algorithm provided in [34, 10].

Using the available bandwidth as cost metric requires us to modify also the local objective function for computing BR wirings. In particular, the best response for v_i may be based on a wiring s_i that maximizes the aggregate bandwidth out of a node given by

$$\sum_{v_j \in V_{-i}} \max_{w \in s} \min\left(AvailBW(e(v_i, w)), AvailBW(w, v_j)\right)$$

The above objective calls for the maximization of the average of the bottleneck bandwidths to all destinations. In [34] we show that finding a local wiring s_i that maximizes this objective function is an NP-hard problem. Thus in our implementation we used a fast local-search heuristic that we verified to be within 5% of optimal in the tested scenarios.[4] Notice that it is straightforward to use the above definitions to produce alternative formulations, *e.g.*, consider the maximization of the *minimum* of the bottleneck bandwidths to all destinations [35].

4.2 Baseline Experimental Results

In this section, we present performance results obtained through measurement of EGOIST . These results allow us to make comparisons between the various neighbor selection policies described in Section 3.2 for the various cost metrics described above. All the results in this section assume that node churn is not an issue – *i.e.*, once it joins the overlay, a node does not leave. Results showing the impact of node churn on EGOIST performance are presented in Section 4.4.

Experimental Setting: We deployed EGOIST on $n = 50$ PlanetLab nodes (30 in North America, 11 in Europe, 7 in Asia, 1 in South America, and 1 in Ocea-

[4] We also added a high penalty when a node is not reachable to guarantee connectedness.

nia). Each of these nodes is configured to recompute its wiring every wiring epoch $T = 60$ seconds. EGOIST nodes are not synchronized, thus on average a re-wiring by some EGOIST node occurs every $T/n = 1.2$ seconds. Whether a node ends up re-wiring or not depends on the neighbor selection policy. For k-Random and k-Regular policies, and since our baseline experiments do not feature any node churn, it follows that these policies will not exhibit any re-wiring. For k-Closest, re-wiring would only be the result of dynamic changes in PlanetLab that result in changes to the cost metric in use (and hence what constitutes the closest set of neighbors). For BR, a node may rewire due to changes in PlanetLab conditions, but may also rewire simply as a result of another node's re-wiring. While in theory [17, 19], BR strategies (under certain conditions) converge to some equilibrium in the Nash sense, we note that this is not likely to be the case for real systems such as EGOIST , since dynamic changes of the underlying system (changes in link delays, bandwidth, and node load) are likely to result in perpetual re-wiring by EGOIST nodes. Setting the wiring epoch T in EGOIST has the effect of controlling the timescale of, and consequently the overhead incurred by, BR re-wiring.

Each experiment presented in this section reflects the results obtained by running EGOIST for at least 10 hours on PlanetLab on January 5th, January 15th, September 15th, October 3rd 2007, and April-June 2008.

Performance Metric: To be able to compare the impact of neighbor selection on the quality of the resulting overlay, throughout this paper we use the *routing cost* (for an individual node or averaged over all nodes) as the main performance metric. For each experiment, an individual cost metric is calculated for every one of the $n = 50$ nodes in the system. The individual cost metric for a node reflects the cost of routing from that node to all other 49 nodes in the system, assuming a uniform routing preference over all destinations.

To facilitate comparisons between various neighbor selection strategies, we often report the *normalized routing cost* (and the 95^{th}-percentile confidence interval), which is the ratio of the cost achievable using a given strategy to that achievable using BR.

Control Variables: In our first set of experiments, our aim is to identify for the three metrics of interest the payoff (if any) from adopting a selfish neighbor selection strategy, *i.e.*, using a BR policy in EGOIST . This payoff will depend on many variables. While some of these variables are *not* within our control (*e.g.*, the dynamic nature of the Internet as reflected by variability in observed PlanetLab conditions), others are within our control, *e.g.*, n, T, and the various settings for our active measurement techniques.

In order to neutralize the effect of extrinsic variables that are not within our control, experiments reporting on different neighbor selection policies were conducted *concurrently.* To do so, we deploy concurrent EGOIST agents on each of the $n = 50$ PlanetLab nodes we use in our experiments, with each agent using a different selection strategy. In effect, each experiment compares the performance of a *set* of concurrently deployed EGOIST overlay networks, each resulting from the use of a particular neighbor selection policy.

One control variable that is particularly important is the *number of direct neighbors*, k, that an EGOIST node is allowed to have. In many ways, k puts a premium on the significance of making a judicious choice of neighbors. For small values of k, choosing the right set of neighbors has the potential of making a bigger impact on performance, when compared to the impact for larger values of k. Thus, in all the results we present in this section, we show the performance of the various policies over a range of k values.

Overview of Performance Results: Before presenting specific performance results, we make two broad observations: first, in all of our experiments, using a BR policy in EGOIST consistently yields the best performance. While such an outcome was anticipated by virtue of findings reported in [19] for a static setting, the results we present here are significant because they underscore the payoff in a *real* deployment, where the modeling assumptions made in prior work do not hold. Second, in all of our experiments, with the exception of BR, no single neighbor selection policy was consistently better than all others across all metrics. In other words, while the performance of a given policy may approach that of BR for one metric while dominating all other policies, such policy dominance does not hold across all the metrics we considered.

Results for Delay Metric: Fig. 1 shows the performance of the various neighbor selection policies in EGOIST normalized with respect to that achievable using BR when the metric of interest is the overlay link/path delay over a range of values for k (with link delays measured using `ping` in the top-left plot, and using `pyxida` in the top-right plot). These results show that BR outperforms all the other wiring policies, especially when k is small, as anticipated in our discussion of the significance of k as a control variable. For example, for $k = 2$, the average delay experienced by an individual node could be anywhere between 200% and 400% *higher* than that achievable using BR. The performance advantage of BR in terms of routing delay stands, even for a moderate number of neighbors. For example, for $k = 5$, BR cuts the routing delay almost by half.

These results confirm the superiority of BR relative to other policies, but do not give us a feel for how close is the performance of EGOIST using BR wiring

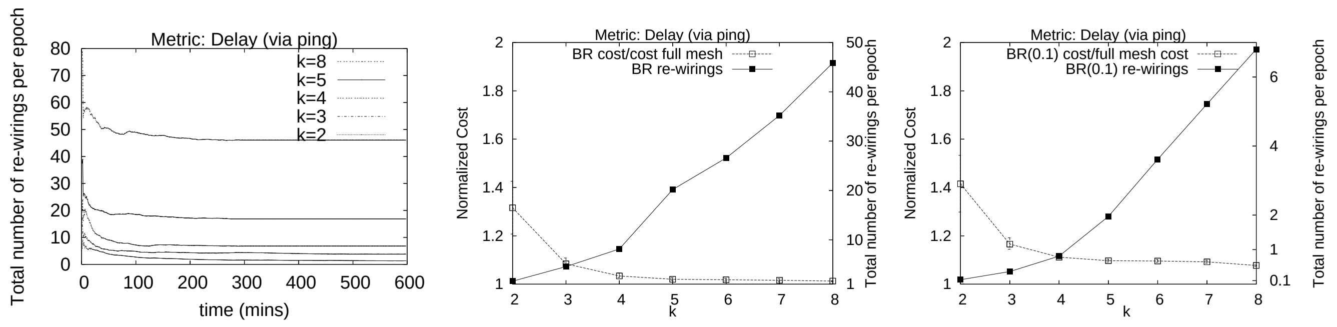

Figure 2: PlanetLab experiments showing the total number of re-wirings per epoch in the system (left), and the relationship between individual cost and total number of re-wirings per epoch in the system with exact best response and an approximate best response with $\epsilon = 10\%$ (center and right respectively), as a function of the number of neighbors k in our EGOIST overlay.

to the "best possible" performance. To do so, we note that by allowing nodes to connect to all other nodes in the overlay, we would be creating a complete overlay graph with $O(n^2)$ overlay links, obviating the need for a neighbor selection policy. Clearly, the performance of routing over such a rich overlay network gives us an *upper bound* on the achievable performance, and a lower bound on the delay metric. Thus, to provide a point of reference for the performance numbers we presented above, in the top-left plot in Fig. 1 we also show the performance achieved by deploying EGOIST and setting $k = n - 1$. Here we should note that this lower bound on delay is what a system such as RON [1] would yield, given that routing in RON is done over shortest paths established over a full mesh, and assuming that any of the $O(n^2)$ overlay links could be used for routing. These results show that using BR in EGOIST yields a performance that is quite competitive with RON's lower bound. As expected, the difference is most pronounced for the smallest k we considered—namely, the lowest delay achievable using 49 overlay links per node is only 30% lower than that achievable using BR with 2 overlay links per node. BR is almost indistinguishable from the lower bound for slightly larger values of k (*e.g.*, $k = 4$).

With respect to the other heuristics, the results in the top plots in Fig. 1 show that k-Closest outperforms k-Random when k is small, but that k-Random ends up outperforming k-Closest for slightly larger values of k. This can be explained by noting that k-Random ends up creating graphs with much smaller diameters than the grid-like graphs resulting from the use of k-Closest, especially as k gets larger. In all experiments, k-Regular performed the worst.

Results for Node Load: The bottom-left plot in Fig. 1 shows the results we obtained using the node load metric, where the path cost is the sum of the loads of all nodes in the path. These results show clear delineations, with BR delivering the best performance over all values of k, k-Random delivering the second-best performance, and k-Closest delivering the worst performance as it fails to predict anything beyond the immediate neighbor, especially in light of the high variance in node load on PlanetLab.

Results for Available Bandwidth: The bottom-right plot in Fig. 1 shows the results we obtained using available bandwidth as the cost metric. Recall that, here, the objective is to get the highest possible *aggregate* bandwidth to all destinations (again, assuming a uniform preference for all destinations) – thus, larger is better. These results show trends that are quite similar to those obtained for the delay metric, with BR outperforming all other policies—delivering a two-fold to four-fold improvement over the other policies, over a wide range of values of k.

4.3 Measurement and Re-wiring Overheads

In this section we show experimentally that EGOIST introduces a rather small amount of overhead for maintaining the overlay structure.

Active Measurement Load: As mentioned in Section 4.2, in the absence of node churn, k-Random and k-Regular do no perform any re-wirings, and thus do not introduce measurement overheads. For k-Closest and BR, the active measurement load is identical.

When the cost metric is delay via `ping`, `ICMP` messages of size 320 bits each (`ECHO` requests/replies) are exchanged once per wiring epoch T. Notice that for established links, there is no need for active measurements since the cost metric for a link would be available by virtue of its use. Thus, the overhead is $\approx (n - k - 1) \cdot 320/T$ bps per node. Using `pyxida`, a single (`http`) request/reply to the `pyxida` server yields the (virtual coordinate space) distances between the node initiating the request and all other nodes in the overlay. This is clearly more efficient than using `ping`, as it injects $\approx (320 + 32n)/T$ bps per node.When the metric is system load, there is no overhead imposed on the network as the system load is measured locally at each node. Finally, when the metric is available bandwidth, our experimental results showed that the bandwidth needed for accurate probing of available bandwidth between two nodes in the overlay is less than 2% of the

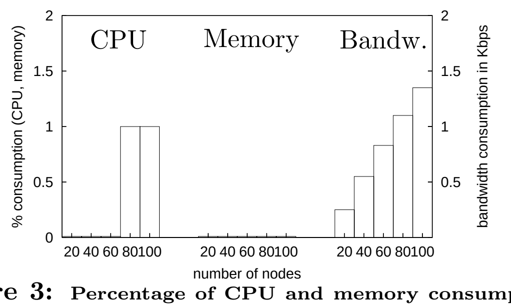

Figure 3: Percentage of CPU and memory consumption, and bandwidth consumption in EGOIST. The metric is delay via `ping`.

pairwise bandwdith.

Link-State Protocol Load: The overhead (in terms of additional injected traffic) imposed by the link-state protocol is also low. Each node broadcasts a packet with its ID, its neighbors' IDs and the cost of the established links to its k neighbors every $T_{\text{announce}} < T$. The header and padding of the link-state protocol messages require a total of 192 bits, and the payload per neighbor requires 32 bits. Thus, the overhead in terms of injected traffic on the overlay is $\approx (192 + 32 \cdot k)/T_{\text{announce}}$ bps per node. In our experiments we set T_{announce}=20 secs. The above can be seen as an upper limit, as only unique link state messages forwarded in the overlay (as mentioned in Section 3.1). In our implementation, no node spent more than 1 Kbps to maintain the network.

Re-wirings Overhead: Fig. 2 (left) shows the total number of re-wirings per (one minute) epoch for the entire overlay over time. The results suggest that the re-wiring rate decreases fast as EGOIST reaches a "steady state" and that the re-wiring rate is minimal for small values of k. Here we note that as k increases the re-wiring rate increases, but the improvement (in terms of routing cost) is marginal, as a small number of outgoing links is sufficient to significantly decrease the cost. This is evident in Fig. 2 (center). Finally, we also note that the re-wiring rate can significantly be decreased (with marginal impact on routing cost) by requiring that re-wiring be performed only if connecting to the "new" set of neighbors would improve the local cost to the node by more than a given threshold ϵ. We refer to this modified version of BR as BR(ϵ). Fig. 2 (right) confirms this by showing the number of re-wirings and resulting performance when $\epsilon = 10\%$.

We measured also the memory and CPU consumption using `time` of Unix. The average CPU and memory utilization was close to 0%. In Fig. 3 we show the average CPU and memory utilization, along with the average bandwidth consumption to maintain the overlay per node. CPU and memory consumption is close to 0%, and the bandwidth consumption per node is negligible. It is worth mentioning that the in-degree was quite uniform in all our experiments, thus no node allocated significantly more CPU power, memory, or bandwidth than any other in the overlay. To improve the scalability even further we have developed a *topology-based biased sampling technique* described and evaluated in [34].

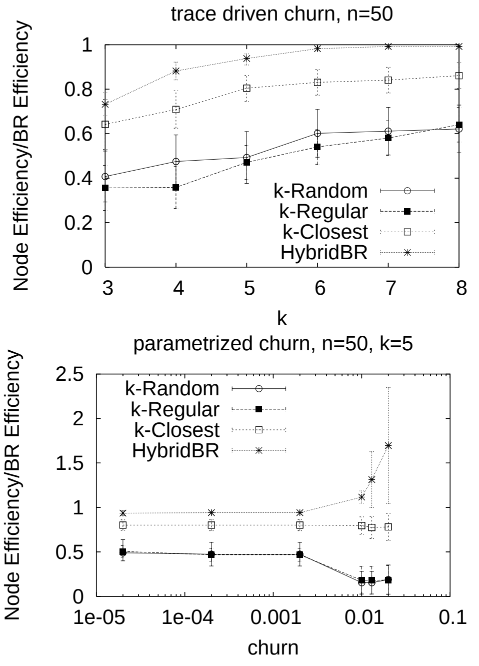

Figure 4: PlanetLab experiments with node churn showing the efficiency of neighbor selection policies (normalized with respect to BR) as a function of the number of neighbors k (top) and churn (bottom) for a 50-node EGOIST overlay.

4.4 Effect of Churn

In the original SNS formulation [19, 17], the graphs resulting from the SNS-game as well as from the empirical wiring strategies were guaranteed to be connected, so they could be compared in terms of average or maximum distance. Node churn, however, can lead to disconnected graphs, therefore we have to use a different metric. For that purpose, we choose the *Efficiency* metric, where the Efficiency ϵ_{ij} between node i and j ($j \neq i$) is inversely proportional to the shortest communication distance d_{ij} when i and j are connected. If there is no path in the graph between node i and j then $\epsilon_{ij} = 0$. The Efficiency ϵ_i of a node i defined as:$\epsilon_i = \frac{1}{n-1} \sum_{j \neq i} \epsilon_{ij}$

To evaluate the efficiency of nodes in EGOIST overlays under churn, we allow each of the $n = 50$ nodes in the overlays to exhibit ON and OFF periods. During its ON periods, a node "joins" the overlay, performs re-wiring according to the chosen policy, and fully participates in the link-state routing protocol. During its OFF periods, a node simply drops out from any activity related to the overlay. The ON/OFF periods we use in our experiments are derived from real data sets of the churn observed for PlanetLab nodes [14], with adjustments to the timescale to control the intensity of churn.

In addition to evaluating the efficiency of various neighbor selection policies we have considered so far, we also evaluate the efficiency of HybridBR (see Section 3.3), which allows a node to donate $k_2 = 2$ of its links to ensure connectivity (*i.e.*, boost the efficiency of the overlay) while using BR for the remaining links.

The top plot in Fig. 4 shows the achievable efficiency of the various neighbor selection policies when churn is present. As before, the efficiency of the various policies is normalized with respect to that achievable using BR, and is shown as a function of k. As with all the metrics we considered so far, BR outperforms all other policies (including HybridBR), but as Egoist nodes are allowed to have more neighbors (*i.e.*, as k increases), the efficiency of the HybridBR approaches that of BR, with the efficiency of k-Closest decisively better than k-Random and k-Regular.

The above results imply that under the level of churn in these experiments, it is not justifiable for BR to donate two of its links simply to ensure connectivity, especially when k is small. Notice that BR overlays that get disconnected due to churn will naturally heal as soon as any of its active nodes decides to rewire. This is so because the (infinite) cost of reaching the disconnected nodes will act as an incentive for nodes to choose disconnected nodes as direct neighbors, thus reconnecting the overlay. As noted earlier, re-wiring occurs every T/n units of time on average (1.2 seconds under our settings), which implies that the vulnerability of BR to disconnections due to churn is highest for smaller overlays and if re-wiring is done infrequently.

Our last question then is whether at much higher churn rates, it is the case that the use of HybridBR would be justified. To answer this question, we changed the timescale of the ON/OFF churn processes to emulate more frequent joins and leaves. The bottom plot in Fig. 4 shows the results by plotting the efficiency metric for the various policies as a function of the churn rate (on the x-axis), which we define (as in [14]) to be the sum of the fraction of the overlay network nodes that changed state (ON/OFF), normalized by time T: Churn= $\frac{1}{T}\sum_{events\ i}\frac{|U_{i-1}\ominus U_i|}{max\{|U_{i-1}|,|U_i|\}}$, where U_i is the new set of nodes in the overlay following an event i that alters the membership in the set of nodes that participate in the overlay, and $\ominus$ is the symmetric set difference. Thus, a churn rate of 0.01 implies that, on average, 1% of the nodes join or leave the overlay per second. For an overlay of size $n = 50$, this translates to a join or leave event every two seconds.

As expected, when churn rate increases significantly to the point where the average time between churn events approaches T/n, the efficiency of HybridBR eventually surpasses that of BR. The results also suggest that under such conditions, the efficiency of both k-Random and k-Regular fall dramatically, whereas that of k-Closest remains level with that of BR.

4.5 Vulnerability to Cheaters

As we discussed in section 3.4, cheating nodes may attempt to game the system by declaring false link costs to their neighbors in order to benefit from Egoist without contributing their own resources to the overlay. Due to the combinatorial nature of the optimization problem underlying BR re-wiring, and the out of order rewirings of individual nodes, it is very hard for individual cheaters to derive the proper costs that will lead to wirings that will be of benefit to them individually, while harming others.Theoretical results [3] advocate that such behavior may even lead to worse equilibria for cheaters in routing games. Thus, in this section, we present results from a series of experiments aimed to assess Egoist's vulnerability to cheaters that misrepresent the cost of their outgoing links (simply by inflating them), in the hopes of discouraging others from selecting them as neighbors.

As described in Section 3.4, one could add mechanisms to detect when cheaters make such false representations. These mechanisms would take the form of passive or active measurement audits from other nodes. Determining how often nodes should perform such random audits and what these nodes do when cheating nodes are identified can be complex. Thus, it would be preferable if one can show that the impact from such abuse is minimal. Clearly, an assessment of the impact of the full spectrum of possible false announcements is beyond the scope of this paper. Thus, we only consider the impact from inflated delay announcements by a single node and by a variable fraction of the nodes.

In Fig. 5 (top), we show the impact from a single cheater announcing link costs that are twice as high as the real ones. The figure shows the individual cost for both the cheater and for all other normal nodes for different values of k. The cost for both types of nodes is very close to the cost without the presence of the cheater. We also evaluate the robustness of Egoist in the presence of many cheaters (up to one-third of the population). These results, shown in Fig. 5 (bottom), yield consistent observations even when the number of outgoing links is very small ($k = 2$), which is the setting in which the impact of bad re-wirings is amplified. These results provide evidence that Egoist is fairly robust to abuse by cheaters, even without the deployment of auditing mechanisms.

5. DISCUSSION AND APPLICATIONS

Egoist is a general purpose overlay routing network that can be used by applications to supplement traditional IP routing. The main difference between

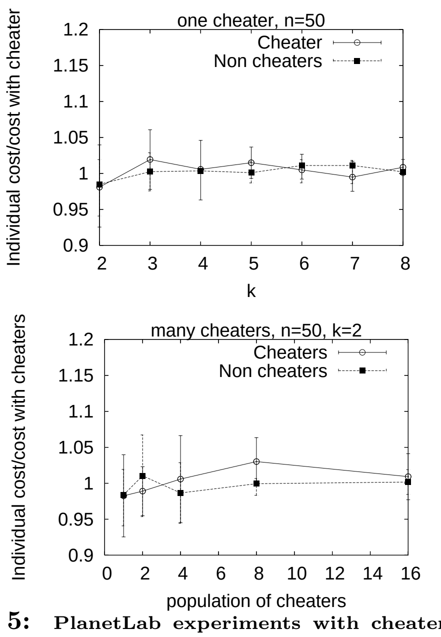

Figure 5: PlanetLab experiments with cheater showing the robustness of neighbor selection policies (normalized with respect to BR) as a function of the number of neighbors k in the presence of one cheater (top) and many cheaters for $k = 2$ (bottom) for a 50-node EGOIST overlay.

an EGOIST overlay and other routing overlays is that by virtue of its BR-wiring strategy, an application contacting its local EGOIST node can be assured that this node will provide better paths than a node that connects to the overlay non-selfishly, *e.g.*, using previously-mentioned random or myopic heuristics. Stated otherwise, the selfish selection of neighbors in EGOIST is just a manifestation of the desire of local applications to get the best possible service for themselves.

An application can connect to an EGOIST node by using a protocol interface that the latter exposes. This is an example of an application instance using EGOIST as a virtual router to communicate over the overlay with another application instance getting access to the overlay from a different EGOIST node. In the artifacts section we provide information on how to access our publicly available implementation which permits using PlanetLab nodes as such virtual routers.

A second option is to integrate EGOIST directly into an application through an API and a corresponding library which we have implemented and made available. In this case, both the application and its local EGOIST instance run at the same node. We have evaluated the performance benefits that EGOIST offers to different kinds of applications, including multi-path file transfer, real-time voice over IP, and multiplayer p2p games. Due to lack of space, we present here some results only for the last one (see [34] for more).

5.1 Multiplayer P2P Gaming on top of EGOIST

Recently there has been intense interest [4, 5] for porting online multi-player games into P2P architectures that scale better and do not require dedicated expensive infrastructure. In this section we demonstrate the potential value of EGOIST for such applications.

We obtained from [5] a trace containing the movements of 100 players (AI bots) participating in a game of Quake III. In Quake III, players are located in a virtual 3D world and interact frequently as they come into contact to fight each other. Two common events of the game are the creation of a new object (*e.g.* a missile), and the update of an existing object (*e.g.* update of its coordinates). Each update is about 230 bytes. All these updates have to be delivered to all the players that are in the vicinity of the affected object in the virtual world. This requires for building a multicast tree rooted at each player that is updating some of its objects.

We distributed the 100 players among our 25 EGOIST nodes on PlanetLab and used the EGOIST overlay to deliver the updates. We set $k = 2$ and mapped the L_3 distance of players i and j in the virtual world into the preference weight p_{ij} that defines the preference that the local EGOIST node of i has for sending messages to the local EGOIST node of j. Since the main requirement in this case is for high interactivity, we employed the delay-based version of BR. With this mapping, nodes pick as neighbors other nodes that host players that are closer in the virtual world which implies interaction, and thus requirement for small end-to-end delay. The value $k = 2$ is justified from the fact that due to human perceptual limitations, players usually pay attention and interact with a small number of other players [5].

In the above setting, we replayed the trace for a period of three minutes involving more than 108,000 events. We compared the update latencies when sent over EGOIST and over k-Random, k-Regular and k-Closest wiring policies. The cumulative distribution function of update latencies is illustrated in Fig. 6. Both the median ($\sim$65 msecs) and the 95^{th}-percentile update latency over EGOIST is less than half of the corresponding latencies over k-Random and k-Regular, and less than two-thirds of those over k-Closest. Experimentally, it has been shown that update latency higher than 200 msecs may effect the quality of user's experience [5]. More than 90% of packets sent over EGOIST were delivered earlier than 200 msecs and only 60-70% under the other topologies.

6. EGOIST ARTIFACTS

Our EGOIST prototype is currently deployed on PlanetLab. A live demonstration of the overlay routing topology maintained by EGOIST can be accessed from the EGOIST project web site at `http://csr.bu.edu/sns/`. We also provide a layered architecture that end users can use in order to route their packets using EGOIST.

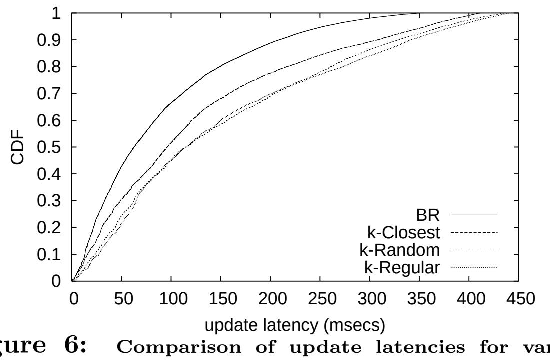

Figure 6: Comparison of update latencies for various neighbor selection policies.

Upon publication of this work, EGOIST source code (in `python`) and the traces from all the experiments used in this paper, will be released to the research community.

7. CONCLUSION

In this paper we have shown how recent theoretical results on Selfish Neighbor Selection could be leveraged for overlay routing applications. Through the development and deployment of our EGOIST prototype routing network on PlanetLab, we have established that Best-Response neighbor selection strategies can indeed be realized in practice, that they provide a substantial performance boost when compared to currently used heuristics, and that they scale much better than full-mesh approaches which require intensive monitoring of $O(n^2)$ links. We have substantiated these benefits under different performance metrics, active and passive link monitoring strategies, in static and churn-prone environments, and in the presence of truthful and untruthful nodes. We also demonstrated that EGOIST incurs minimal overhead and how it can be used as a building block for efficient routing in overlay applications.

8. REFERENCES

[1] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *SOSP '01*.

[2] V. Arya, N. Garg, R. Khandekar, K. Munagala, and V. Pandit. Local search heuristic for k-median and facility location problems. In *Proc. of ACM STOC '01*.

[3] M. Babaioff, R. Kleinberg, and C. H. Papadimitriou. Congestion games with malicious players. In *EC*, 2007.

[4] A. Bharambe, J. Pang, and S. Seshan. Colyseus: a distributed architecture for online multiplayer games. In *NSDI '06*.

[5] A. Bharambe, J. Pang, and S. Seshan. Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games. In *SIGCOMM '08*.

[6] A. Blanc, Y.-K. Liu, A. Vahdat, and S. Shenker. Designing incentives for peer-to-peer routing. In *P2PEcon '04*.

[7] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI '02*.

[8] B.-G. Chun, R. Fonseca, I. Stoica, and J. Kubiatowicz. Characterizing selfishly constructed overlay routing networks. In *INFOCOM '04*.

[9] J. Corbo and D. C. Parkes. The price of selfish behavior in bilateral network formation. In *PODC '05*.

[10] N. Deo. *Graph Theory with Applications to Engineering and Computer Science*. Prentice Hall, 1994.

[11] O. Ercetin and L. Tassiulas. Market-based resource allocation for content delivery in the internet. *IEEE Transactions on Computers*, 52(12):1573–1585, 2003.

[12] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a network creation game. In *PODC '03*.

[13] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. In *EC '04*.

[14] P. B. Godfrey, S. Shenker, and I. Stoica. Minimizing Churn in Distributed Systems. In *SIGCOMM '06*.

[15] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03*.

[16] J. Han, D. Watson, and F. Jahanian. Topology aware overlay networks. In *INFOCOM '05*.

[17] N. Laoutaris, L. Poplawski, R. Rajaraman, R. Sundaram, and S.-H. Teng. A Bounded-Degree Network Formation Game. In *PODC '08*.

[18] N. Laoutaris, P. Rodriguez, and L. Massoulie. Echos: edge capacity hosting overlays of nano data centers. *SIGCOMM Comput. Commun. Rev.*, 38(1):51–54, 2008.

[19] N. Laoutaris, G. Smaragdakis, A. Bestavros, and J. Byers. Implications of Selfish Neighbor Selection in Overlay Networks. In *INFOCOM '07*.

[20] J. Ledlie, P. Pietzuch, and M. Seltzer. Network Coordinates in the Wild. In *NSDI '07*, 2007.

[21] Z. Li and P. Mohapatra. Impact of Topology On Overlay Routing Service. In *INFOCOM '04*.

[22] Z. Li and P. Mohapatra. QRON: QoS-aware routing in overlay networks. *IEEE JSAC*, 22(1):29–40, Jan 2004.

[23] Y. Liu, H. Zhang, W. Gong, and D. F. Towsley. On the interaction between overlay routing and underlay routing. In *INFOCOM '05*.

[24] G. S. Manku, M. Naor, and U. Wieder. Know thy neighbor's neighbor: the power of lookahead in randomized P2P networks. In *STOC '04*.

[25] T. Moscibroda, S. Schmid, and R. Wattenhofer. On the topologies formed by selfish peers. In *PODC '06*.

[26] L. Qiu, Y. R. Yang, Y. Zhang, and S. Shenker. On Selfish Routing in Internet-like Environments. In *SIGCOMM 03*.

[27] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically aware overlay construction and server selection. In *INFOCOM '02*.

[28] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *USENIX '04*.

[29] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *PAM '03*.

[30] T. Roughgarden and E. Tardos. How bad is selfish routing? *J. ACM*, 49(2):236–259, 2002.

[31] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: Informed Internet routing and transport. *IEEE Micro*, 19(1):50–59, 1999.

[32] S. Seetharaman and M. Ammar. On the Interaction between Dynamic Routing in the Overlay and Native Layers. In *INFOCOM '06*.

[33] A. Shriram, M. Murray, Y. Hyun, N. Brownlee, A. Broido, M. Fomenkov, and K. C. Claffy. Comparison of Public End-to-End Bandwidth Estimation Tools on High-Speed Links. In *PAM '05*.

[34] G. Smaragdakis, N. Laoutaris, A. Bestavros, J. Byers, and M. Roussopoulos. EGOIST: Overlay Routing using Selfish Neighbor Selection. Technical Report BUCS-TR-2007-013, CS Department, Boston University.

[35] G. Smaragdakis, N. Laoutaris, P. Michiardi, A. Bestavros, J. W. Byers, and M. Roussopoulos. Swarming on Optimized Graphs for n-way Broadcast. In *INFOCOM '08*.

[36] Srinivasan Seetharaman and Voler Hilt and Markus Hofmann and Mostafa Ammar. Preemptive strategies to improve routing performance of native and overlay layers. In *INFOCOM '07*.

[37] V. Vishnumurthy and P. Francis. A comparison of structured and unstructured p2p approaches to

heterogeneous random peer selection. In *USENIX '07*.

[38] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson. Reliability and Security in the CoDeeN Content Distribution Network. In *USENIX '04*.

[39] Z. Yao, D. Leonard, X. Wang, and D. Loguinov. Modeling Heterogeneous User Churn and Local Resilience of Unstructured P2P Networks. In *ICNP '06*.

[40] A. Young, J. Chen, Z. Ma, A. Krishnamurthy, L. L. Peterson, and R. Wang. Overlay Mesh Construction Using Interleaved Spanning Trees. In *INFOCOM '04*.

[41] Y. Zhu, C. Dovrolis, and M. H. Ammar. Dynamic overlay routing based on available bandwidth estimation: A simulation study. *Computer Networks*, 50(6):742–762, 2006.