Altruistic Routing in Structured P2P Networks

Sukumar Ghosh   Alina Bejan   Amlan Bhattacharya
{ghosh, abejan, bhattach}@cs.uiowa.edu

Department of Computer Science
University of Iowa

Abstract

Request patterns in P2P networks are not uniform, and the cost of communication depends on the traffic flows among peers. This paper illustrates how the processes in an overlay network can use the information about traffic flow pattern to modify their routing tables and minimize their communication costs. Two different adaptation strategies: *selfish* and *altruistic* are described and analyzed. The *selfish* protocol modifies the routing tables to suit each process individual needs, and is easy to implement, but the improvements are limited. Compared to this, the *altruistic* protocol that adjusts routing tables based on the needs of *other* processes, promises a much better performance. Experimental results support this observation. Despite the promise, there are concerns about the applicability of the altruistic protocol in an untrusted P2P network environment.

Keywords and Phrases. P2P network, adpative overlay network, flow adaptation, route caching, selfish and altruistic policies.

1 Introduction

Motivation. A P2P network is an Internet-based distributed system for the efficient and scalable location of remote objects without any central authority. The past few years have witnessed a phenomenal growth of these network targeted towards a variety of applications. For all such networks, efficient routing is a key concern. We focus on structured P2P networks that use distributed hash tables (DHT).

Two metrics of performance of DHT-based P2P networks are the *space complexity* for the individual nodes, and the *time complexity* for searching remote objects - both should be as small as possible. The space complexity includes the space needed to store the routing table, as well as the time needed to store replicas of objects. The time complexity, on the other hand, translates to hop count if geographical proximity between nodes is disregarded. The space and time complexities, in general, conflict with one another. A sparse interconnection network like a ring has a constant size routing table, but the routing distance for queries may be as large as $O(n)$, whereas for a completely connected topology all queries are resolved in a single hop, but at the expense of a routing table of size $O(n)$. Existing P2P like CAN [7], Chord [10], Pastry [9], Tapestry [12] use interconnection topologies and routing mechanisms that strike a balance between these two extremes.

Despite providing good solutions for meeting the conflicting goals of *fast lookups* and *small number* of states per process, these networks provide a performance that is immune to variations in the traffic flow. In real life however, such variations do exist. We quantify performance as follows: Let $f(i, j)$ denote the *flow* from i to j (measured by the number of packets delivered per unit time), and $l(i, j)$ denote the latency (as measured by the number of hops) needed for one such communication. Two different types of cost functions may be used as yardsticks of performance:

Private cost. For each process i, $C_i = \sum_j f(i, j).l(i, j)$, and

Total cost. $C = \sum_{i,j} f(i, j).l(i, j) = \sum_{i=1}^{n} C_i$.

The above two costs interplay in different ways. While a low value of the total cost reflects an efficient network, individual processes may have a competitive goal, much in the spirit of games. Indeed, a process that is responsible for routing the communications between other processes may try to use a routing policy that minimizes its own cost, possibly at the expense of others. Such market based approaches are being studied in internet routing. For example, in the *Wardrop model* of routing, Roughgarden and Tardos [11] proved the important result that *selfish routing* policies (where individual processes are free to pick a routing policy that serves their self-interests) increase the routing cost by only a constant factor from the optimal cost. The Wardrop model is defined by a tuple (G, r, l), where $G = (V, E)$ defines the network topology, $r : E \to \aleph$ represents the network flow where $\aleph$ denores the set of non-negative integers, and the edge latency l is a non-decreasing function of the flow through an edge.

Our model and problem are different. Given a flow between every pair of nodes, our goal is to minimize the communication cost by modifying the *routing tables* of the individual processes. We propose to augment each routing table using a small predefined number of cache entries, that will accelerate the routing. Thus, routing table reconfiguration will include modification of the routing cache entries. We assume that processes acquire knowledge of the flow using a profiling algorithm that runs in the background. The variables are (1) the routing table entries of the individual nodes, and (2) routing policies (although we restrict ourselves to the well-known DHT-based routing policies or their variations). Given the flow, we want the processes to *self-organize* their routing tables for minimizing the cost functions. The routing tables determine the immediate successor of each nodes in the overlay network. This is an exercise in adaptation, where the network topology is required to adapt to the flow for minimizing the cost functions. Thus, G itself is not constant. Furthermore, compared to the Wardop model, the latency function is simpler in that it only reflects the hop count, and is independent of the flow through the links.

The problem under consideration involves a subtle interplay between cooperation and competition. Processes cooperate with one another to guarantee that eventually every packet reaches its destination. Processes also cooperate when the goal is to minimize of the global cost. The competitive aspect surfaces when the goal is the minimize the private costs: each process may want to minimize its private cost at the expense of other processes. The latter part can be viewed as a non-cooperative game – the target configuration will reflect a Nash equilibrium, where no user can unilaterally decrease its private cost by unilaterally changing its strategy.

Related work. On-demand routing is a common feature in mobile ad-hoc networks [**?**] - the primary issues are reachability and loop-free routing. In classical networks, *flow-adaptivity* has been addressed at different layers. For example, TCP congestion control addresses flow adaptivity, although the solution targets end-to-end control instead of tweaking the network layer. In QoS maintenance and error control, Ludwig [6] proposed a method of service differentiation by tailoring link layer error control to the QoS requirements of flows sharing wireless links. In P2P networks, communication cost reduction via query and result caching has been proposed for Gnutella [**?**]. Wang et al [**?**] described a distributed caching and adaptive search protocol for reducing the search traffic in Gnutella network. In structured networks, Bejan and Ghosh [2] showed how the information about flow patterns can be used to identify clusters of interest, restructure the network to lower the cost of intra-cluster communication, and reduce the average communication cost incurred by the processes. On the same line of using clustering of peers we can cite Triantafillou's work [13] that combines selfishness and altruism at the overlay level. The difference is that altruism refers to the ability/willingness of peers to contribute their resources rather than the willingness of peers to route queries on behalf of other peers, at the expense - sometimes - of increasing the hop count for its own queries. The difference is major, since our selfishness property manifests itself at the routing level, and not at content one. The idea of selfish behavior in P2P systems appears in [**?**] as well, but it concerns replication of resources by peers that behave selfishly. Again the selfishness is reflected at the content level, more precisely the paper studies selfish caching from the game theory perspective. Finally, numerous distributed systems develop adaptation protocols in the application layer for a variety of applications.

Contributions. This paper discusses algorithms for self-optimization using the competitive view, where each process intends to minimize its private cost. We do not consider content replication at the moment - content replication can always be used to further improve performance on top of our proposed methods. We present two approaches. The first method uses a *selfish* version of route caching [1], where the contents of the caches reflect the links with highest flows for the individual processes. The second approach studies a counter-intuitive protocol: *altruistic route caching*, where each route cache stores the highest flow links of *other* processes only, instead of its own. We analyzed and experimented these protocols to improvize query processing on the basic Chord network [10], and demonstrated that altruistic route caching performs much better than its selfish counterpart under light loads. We revisit the concern that in an environment of mutual mistrust, an altruistic policy may be impractical. So to promote altruism (over a selfish protocol), additional measures will be necessary, much in the spirit of coping with the freeloaders problem.

Organization. **Section 2** formalizes the model and the objective of this paper. **Section 3** presents a conservative routing table self-configuration algorithm that guarantees cost minimization for a given flow. **Section 4** introduces the selfish and the altruistic route caching protocols, and analyzes their complexities. **Section 5** describes the simulation results, and proves our thesis. Finally, **Section 6** contains some concluding remarks.

2 Preliminaries

We consider a structured P2P network whose interconnection graph is $G = (V, E)$. Each node of $v \in V$ denotes a process, and each edge $(i, j) \in E$ represents a directed edge

[1]The route cache contains the IP addresses of a small number of peers towards which queries are forwarded using a modified routing algorithm.

from i and j. Each node has m edges connecting to m different peers, where $m << n$. When $m = \lceil \log n \rceil$, node identifiers are hashed into keys ranging from 0 through $n-1$, and routing table entries (also called *fingers*) of node i point to the peers mapped to keys $(i+1), (i+2), \cdots (i+2^{m-1})$, the topology reduces to that of Chord [10]. The routing distance between any pair of nodes is bounded from above by $\lceil \log n \rceil$. We will frequently cite Chord as our *base* network.

Let $f : V \times V \rightarrow \aleph$ denote the network flow, where $\aleph$ is the set of non-negative integers. Given the flow, the network configuration and the routing policy, one can easily compute the total cost of communication in the network. Performance demands these costs to be as low as possible. When the routing tables are static and independent of the flows, the routing cost is not optimal. To realize the feasibility of routing table reconfiguration for optimizing communication cost, consider the following case of Chord: a node i frequently communicates with another node j, but each communication from i to j takes $\log n$ hops. Clearly, the communication cost can be reduced if i sets up a finger directly pointing to j. If this finger replaces an existing finger, then this reconfiguration of the routing table will have impact on other flows in the network. Another possibility is to leave the existing fingers undisturbed, but use a small number k (k is of the order of m) of extra fingers that will serve as a cache that will accelerate the routing. Such optimizations are the cornerstones of our proposal. Sometimes, a reduction in the routing distance between some pair of nodes may come at the expense of an increase in the routing distances between some other pair of nodes which do not frequently communicate with one another. Common engineering practices encourage making the common case faster. Since caches have limited entries, routing caching policies have to deal with conflicting requirements. Given a flow and a small number of cache entries per node, evolving a policy that will redefine routing tables and reduce the overall communication cost makes an interesting problem to study. A policy is acceptable, when it satisfies the following three requirements:

Stabilization. Given a flow f, the routing table reconfiguration algorithm must converge to a final configuration, and continue in that configuration unless the system records a change of flow.

Reachability. The terminal configuration of the routing tables must guarantee that each packet sent by a node reaches its destination within a bounded number of hops.

Cost optimization. The communication cost for the given traffic flow must be as low as possible, certainly lower that what it was for the unoptimized network.

In this paper, we rule out content caching [**?**] (used in Gnutella) in favor of route caching since the latter has much lower space overhead.

2.1 The routing policy

We limit ourselves to a form of route caching where a cache entry is used as the last hop, i.e. a cache hit signals packet delivery to the destination node. For any packet directed towards a desination, if no such entry exists in an intermediate node, then the packet is routed using the existing fingers. As an alternative, one could also use the cache entries to expedite interim progress, but our experiments revealed that it did not have any noticeable impact on accelerated delivery. The routing policy can thus be summarized as follows:

```
if $\exists j \in$ cache : $j$ points to destination
        then use $j$ to route the packet
        else follow the traditional Chord routing
fi
```

Using the above routing policy, the following theorem trivially holds:

Theorem 1. *The number of hops required to route any packet is bounded from above by* $\log n$.

Proof. Until there is a cache hit, the basic routing of Chord is applicable. When a hit occurs, the destination is reached in the next hop. □

The improvements in routing will depend on the content of the routing cache. A large number of hits within a small size cache will lower the communication cost. A clever choice will lead to more cache hits, and reduce the communication cost. In this paper we discuss various route caching strategies to maximize cache hits and accelerate routing.

3 Self-optimization protocols

Any strategy for route reconfiguration must satisfy the requirements of stabilization, reachability, and cost optimization. Theorem 1 establishes reachability, so we will focus on the other two requirements.

A trivial lower bound of the communication cost is $C = \sum_{i,j} f(i,j)$. No doubt this is very loose as it applies to a completely connected graph only. But it serves as a reference with respect to which we can compare the communication cost. Computing a tighter lower bound is an open problem. Note that finding an optimal solution that leads to the minimum communication cost is extremely complex. In [8], Reshef addressed the problem of constructing the *minimum communication cost spanning tree* (MCT) for a weighted graph. The communication cost between any pair of nodes is the sum of weights of all the edges in the path between the two nodes, and the total communication cost is the sum of all such costs for all destination nodes. Reshef showed that even when all weights are 1, the problem is NP-hard. Our problem reduces to the MCT problem when the flow between every pair of nodes is identical.

In this section, we describe two route caching protocols that adapt to the flow, and reduce communication costs.

3.1 A selfish protocol

Let the size of the routing cache be k ($1 \leq k \leq \log n$). There is no rational basis for this except for the fact that it should be small. The essence of the selfish protocol is as follows:

> Each node i sorts the flows to all the other nodes $j \neq i$ in a descending order, and sets its routing fingers in its cache to the top m flow destinations in this order.

Since the actions do not interfere with one another, the routing tables will trivially stabilize after n steps or one round. Although the traffic to the top k destinations will reach in one hop, traffic towards other destinations will not receive any special help from any other node, unless such a destination matches with one of the top k flow destination of a node en route.

Complexity analysis. Measurements reveal that traffic on P2P networks follows a Zipf distribution [3, 4]. We approximate a Zipf distribution by identifying a variable h that defines the number of destination nodes towards the flow is substantially higher than the flows to the remaining nodes. We call these nodes the *top flow destination.* In computing the cost, we will consider these nodes only, and ignore the role of the remaining nodes.

Selfish route caching helps deliver packets to k-out-of-h top flow destinations in a single hop. For the remaining $(h-k)$ destinations, we will calculate the expected number of hops needed before a cache hit occurs.

If the k fingers in the cache are randomly oriented, then the probability that a packet originating from a node i will hit the cache of a neighbor j is $\frac{k}{n-1}$, the latter term reflects the probability that destination of this packet also is one of the top k destinations (stored in the cache) of node j. If this is not true, then the probability that this packet is not delivered in two or fewer hops is $1-\frac{k}{n-1}$. This packet will then check if its destination hits the cache of the next node, and the probability that the packet is delivered in three hops is $\frac{k}{n-1}(1-\frac{k}{n-1})$. Continuing these arguments, it is easy to show that the probability of cache hit after r hops is

$$\frac{k}{n-1}.(1-\frac{k}{n-1})^{r-2}$$

Let $x = 1-\frac{k}{n-1}$. Then the expected number of hops before a cache hit occurs follows a *geometric distribution* $x.(1-x)^{r-1}$.

It follows that the expected number of hops $E(l)$ for a cache hit is $\frac{1}{x}$, which equals $\frac{n-1}{k}$.

When $k \leq \log n$ and n is large, the expected hop count for a cache hit is much greater than $\log n$ since $\frac{n-1}{k} >> \log n$. However, the primary fingers guarantee delivery of every packet within $\log n$ hops. Therefore packets to the remaining $(n-k-1)$ destinations are expected to reach their destinations before any cache hit occurs.

The average hop count for the basic Chord is $\frac{\log n}{2}$. If a fraction α of the total traffic towards the top flow destinations is directed to the top k destinations, then $k = \alpha.h$. The private communication cost will be $h.(\alpha + (1-\alpha).\frac{\log n}{2})$ times the average flow towards a top flow destination.

So the expected improvement in the average hop count with selfish caching is

$$\frac{\alpha + (1-\alpha).\frac{\log n}{2}}{\frac{\log n}{2}}$$

.

For a network with 100,000 nodes, if $h = 200$ and $k = 16$, then $\alpha = 0.08$, and the expected improvement in the average hop count is only 2% which is marginal. To make a difference, the cache size must be larger. In this case, a cache of size 50 will improve the hop count by 28%.

3.2 An altruistic protocol

The altruistic approach is just opposite to the selfish approach:

> Each node i monitors the flows originating from other nodes $j \neq i$ and being routed through i, sorts these flows in the descending order, and sets its routing fingers in its cache to the top k flow destinations in this order.

The altruistic protocol allows a node i to route packets to its top destinations using the routing fingers in the cache of *other nodes* $\{j : j \neq i\}$. The intuition here is that, if a node does not behave in a selfish way, then it will get the support of all other nodes for routing its packets to its top destination.

Complexity analysis. Consider a lightly loaded system, where only a small number of nodes are active, and most traffic is directed to only a few preferred destinations. Furthermore, traffic for each active node in the network follows the Zipf distribution. A packet originating at a node i and directed to a top flow destination will first reach one of its k neighbors as determined by its routing table. Since the contents of its local cache i are determined by flows originating at other nodes $j \neq i$, it is unlikely that the packet will reach its destination in one hop. However, there is a high probability that one of the neighbors of i has a routing cache entry directed to the final destination of this packet. In the absence of any contention this probability is a guarantee. Furthermore, if the top flow destinations are well scattered in the key space, then the packets can make the best use of the routing caches at each destination.

With *no contention*, the number of cache hits after the first hop will range between k and $m.k$ (Fig 1). The lower bound corresponds to the case when all the top flow destinations lie between the nodes pointed by two consecutive fingers of the routing table. The upper bound corresponds to the case in which the top flow destinations are evenly scattered so that between the nodes pointed by two consecutive fingers, there are k such nodes. This estimate disregards the cases in which the separation between the nodes pointed by consecutive fingers is less than k, or one of the existing fingers in the routing table already points to a top flow destination.

Since there are $n.k$ cache entries in the entire system, at most $n.k$ destinations can benefit from cache hits. For flows with a global rank lower than $n.k$ there is no guarantee of a cache hit, but the upper bound of $\log n$ steps will prevail. As more nodes become active, contention grows, and the expected number of hops not only depends on the distribution of the top flow destinations, but also on the ability of the individual nodes to seize a cache entry in a node that falls on the route. If a fraction of the nodes is active, then the routing complexity will depend on how many of the cache entries point to the top destinations of the packets originating from the active nodes.

The altruistic protocol performs particularly well under light loads, since the route caches of the inactive nodes en route the destination expedite the delivery of the packets originating from the active nodes. That said, the stabilization of the routing caches is an

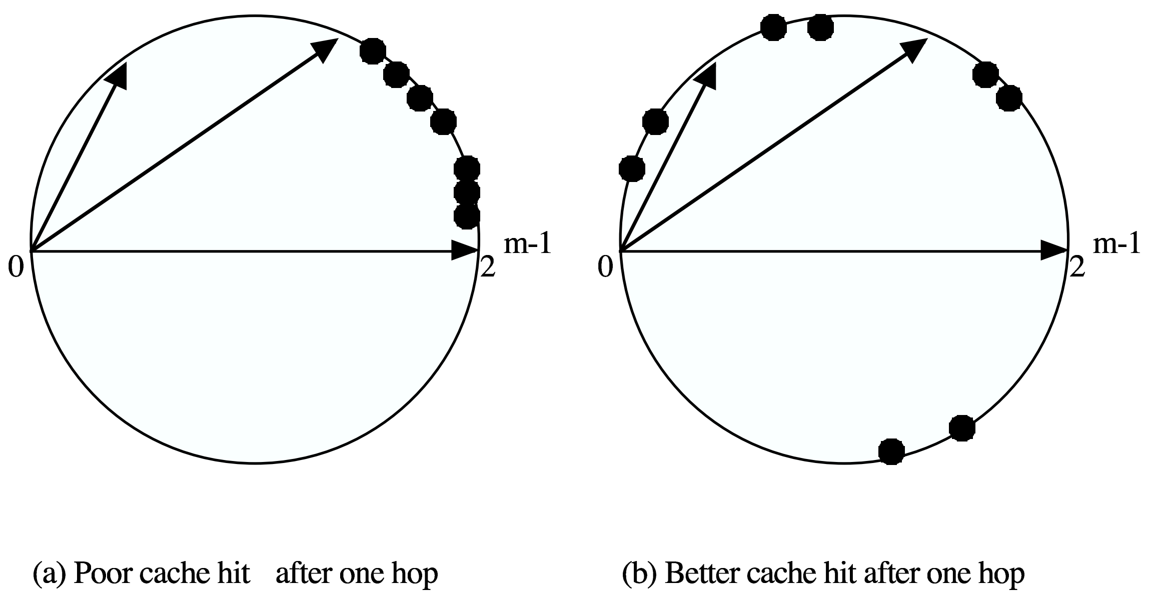
0
2 m-1
0
2 m-1
(a) Poor cache hit after one hop
(b) Better cache hit after one hop

Figure 1: Two cases of cache hits: (a) has much lower hits than (b).

overhead. The intermediate nodes sense and profile all packets routed through them, and modify the routing caches on-the-fly. Unless the routing cache entries stabilize following a change in the flow, it is not feasible to guarantee any performance improvement in packet delivery beyond the lower bound.

Let h be the number of top destinations of a process in a network of size n. We will assume that these are uniformly scattered over the entire key space. The actual value of h relative to n will depend on a parameter α: here we approximate Zipf distribution by a rectangular envelope and assume that only α fraction of the total traffic is directed towards the top destinations – traffic towards the remaining destinations is negligibly small.

We will compute the expected number of hops needed to reach an arbitrary destination (from the set of top destinations) using altruistic caching. Consider *Figure 1b*, and without loss of generality, let process 0 generate a request to one of its top destinations *top*. Depending on the possible location of *top* on the Chord ring, we will compute the probability of reaching *top* in a given number of hops.

Case 1. $2^{n-1} < top < 2^n$

The probability of $2^{n-1} < top < 2^n$ is $\frac{1}{2}$. We will compute the probability of a packet reaching *top* in two hops. Unless $top = 2^{n-1} + 2^j (j < n-1)$ (which has a low probability), routing has to be done using one of the cached fingers of node 2^{n-1}. If there are k fingers available in the cache, then to reach the destination in two hops, the flow has to rank in the top k of all flows routed through that node. The contenders are packets originating from nodes $2^{n-1} - 2^j (j < n-1)$. The fraction of packets originating from a source node $2^{n-1} - 2^q$ and vying for a cache entry is $\frac{h}{2^{n-q}}$. Therefore the number of contenders will be

$\frac{h}{2} + \frac{h}{4} + \frac{h}{8} + \cdots + \frac{h}{2^{n-1}}$

For large networks, this approximates to h. Thus the probability that the flow from node 0 towards *top* will be able to rank in the top k is $\frac{k}{h}$, and the probability that a packet will reach *top* in two hops is $\frac{1}{2}.\frac{k}{h}$.

Case 2. $2^j < top < 2^{j+1}$

Using similar arguments, it is possible to show that $\forall j : 1 \leq j \leq n-1$, the probability that a destination *top*: $2^j < top < 2^{j+1}$ is reached in two hops is $\frac{1}{2^{n-j}}.\frac{k}{h}$. (See Appendix). Since *top* is an arbitrary top destination, the probability of reaching *top* in two hops is

$\frac{k}{h}.(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^{n-1}})$

$= \frac{k}{h}$ (for large n)

Let $p = \frac{k}{h}$. It can be shown that the probability of reaching *top* in r hops is $p.(1-p)^{r-1}$ Thus the hop count is a random variable with a geometric distribution, from which it follows that

$E(\text{hop count to reach } top) = 1 + \frac{h}{k}$

However, the maximum number of hops to any destination cannot exceed $\log n$. So the expected hop count is $min(1 + \frac{h}{k}, \log n)$.

In a P2P network, typically not all nodes are active at the same time. Let β be the fraction of nodes that are active. We will call β the *activity ratio*. When this factor is taken into account, contention for cache entries reduces by a factor β and the expected hop count to reach an arbitrary top destination becomes $min(1 + \frac{\beta h}{k}, \lceil \log n \rceil)$.

Let us compare this with the routing efficiency obtained via selfish route caching. First, the routing complexity of selfish caching does not depend on the activity ratio, since regardless of the traffic, other nodes are not required to cooperate by lending cache fingers. So, if $h >> k$ then traffic towards $(h-k)$ of the h destinations is unlikely to use the cache, and suffer from the average hop complexity of $log(n)$. Assume that $n = 100,000$ (thus $\lceil \log n \rceil = 17$), and $h = 500$. Here $(500 - 17) = 481$ will be reached in approximately 9 hops. However, using altruistic caching, when $\beta = 0.1$, the routing distance is 4 hops. Altrusitic caching thus performs much better when the system is lightly loaded. The performance is expected to get worse when the activity ratio increases.

`We need to include a graph here showing the theoretical results`

4 Flow profiling

The first step in flow-adaptive routing is flow profiling. A naive approach is for each node i to count the number of packets to each destination $j \neq i$, sort them, and set the routing cache entries to the top k flow destinations. This is not acceptable due to space complexity reasons. Counting and recording all flows to every destination will need $O(n)$ space, and affect the scalability, particularly when the routing table takes up only $O(\log n)$ space. We therefore adopt a variation of a caching protocol that performs an approximate sorting of the top k flow destinations using $O(k)$ space. The algorithm is described below.

Consider a buffer of size $O(k)$. For any node, this buffer stores the identifiers of the packet destinations. Initially the buffer is empty. The lifetime of an entry is measured by the *counter* variable, which is refreshed with every new query for that specific key. If a key is not queried for, its *counter* will eventually reach 0 and thus automatically disappers from the fats table. When node i sends out a packet to node j, it executes the following steps:

```
Cache reconfiguration algorithm for node $i$

var FT; // the Fast Table
var current; // the current number of entries in the fast table

if $\exists\, 0 \leq t < current\ : FT[t] == key\ \rightarrow$
    FT[current++] = key;
    FT[current].counter = MAX-COUNTER;
    for i=0 to current-1
            FT[i].counter–;
fi
if $not\ \exists\, 0 \leq t < current\ : FT[t] == key\ :\rightarrow$
    if $\exists\, 0 \leq l < current$ : FT[l].counter ==0
            FT[l] = key;
            FT[l].counter = MAX-COUNTER;
    fi
    for i = 0 to current
            if l $\neq$ i : FT[i].counter –; fi
fi
```

Thus, in case of a buffer hit, the entry is removed from the current location and placed at the front, and the remaining entries are compacted. Each buffer entry corresponds to an entry in the routing cache.

Clearly this algorithm identifies the set of nodes with which a given node has communicated *most recently*. Any node that has not received a communication during tha past *max* steps automatically drops out of the buffer. The flow profiling algorithm for the altruistic protocol works similarly, except that the buffer entries contain destination identifiers of packets originating at other nodes.

Theorem 2. *For a given flow, the routing cache entries stabilize in a bounded number of steps.*

Proof. Per the routing algorithm, cache entries define the *last hop* of route. As a result, while the flow profiling is in progress, the cache entries may change, but each such change does not affect any other route. Thus there is no cyclic dependency. Therefore, in a bounded time after the last flow change has taken place, the cache entries will stabilize. $\square$

5 Simulation results

We conducted experiments on a P2P simulator specifically geared towards Chord. Our simulator could handle a system of size up to 5000 nodes. The request pattern was assumed to follow a **Zipf** distribution with $\alpha = 1.0$. We conducted the experiments with 100000 events, and assumed that join and leave operations account for 1% of the total set of events (the rest being lookups), and the fast table size of $log(n)$. The experiments allowed existing nodes to leave and new nodes to join per the original Chord protocol.

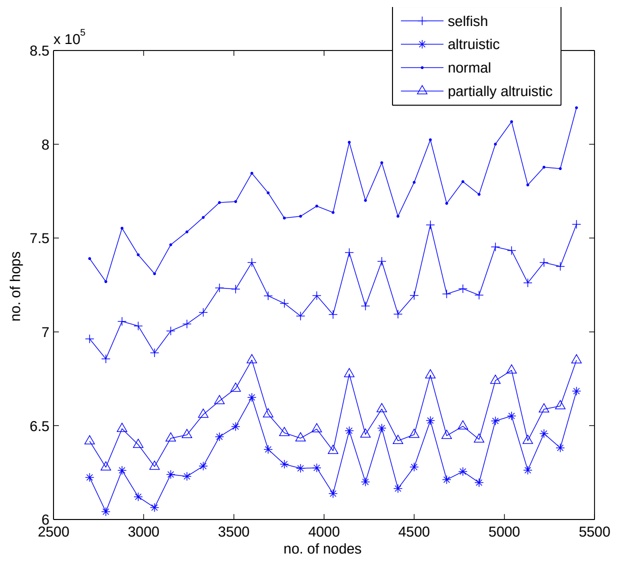

Figure 2: Comparative performance of the four routing algorithms.

Figure 2 shows the improvements achieved by the selfish and the altruistic caching protocols with a cache size of $\log n$ per node. The simulation clearly confirms that altruistic caching leads to faster routing.

Figure 3 and Figure 4 compare the effectiveness of the two caching mechanism by comparing the hit ratios in selfish and altruistic caching.

Figure 5 shows the impact of system load. As expected, the improvements caused by altruistic route caching is much more prominent when the activity ratio is low. The system load does not have any significant impact when selfish caching is used.

6 Conclusion

Our results clearly show that altruistic route caching significantly improves the performance of routing in structured P2P networks. Both selfish and altruistic versions are self-adaptive. Although the altruistic protocol has a better performance, there is some concern about whether this can be adapted in practice. Despite cooperation from most peers, each peer

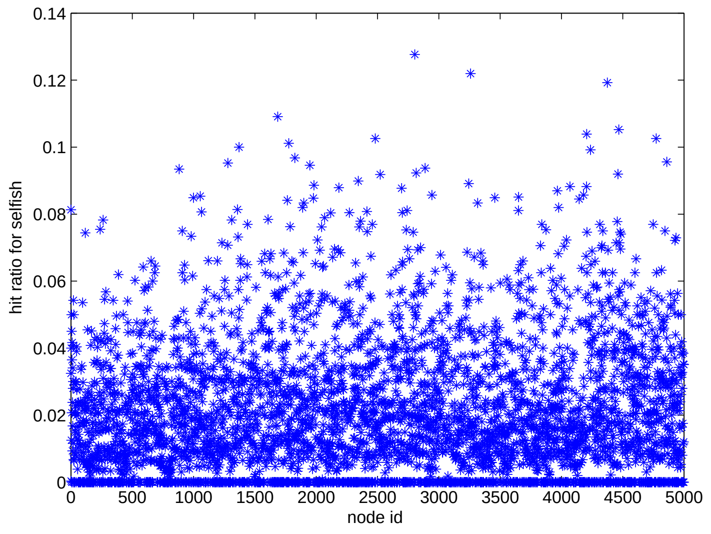

Figure 3: Hit ratio in selfish routing

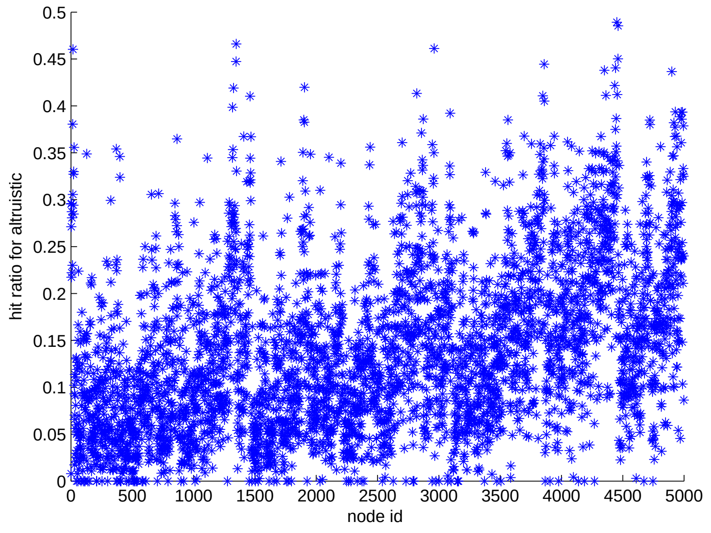

Figure 4: Hit ratio in altruistic routing.

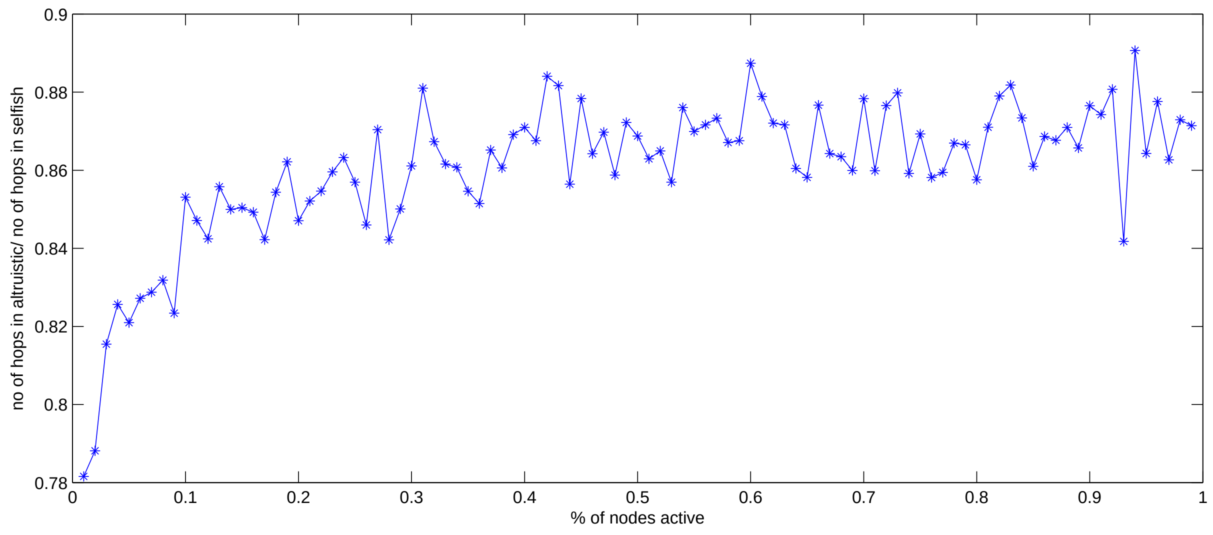

Figure 5: The impact of system load on the relative performance of altruistic and selfish routing

also has some degree of mistrust against a fraction of them, and might wonder: *I am using altruistic caching, but what if others follow the selfish protocol?* In terms of game theory, Nash equilibrium corresponds to every peer using the selfish protocol. This is an inferior but stable equilibrium, since no peer can unilaterally improve its payoff (i.e. performance) by switching to the other protocol. There is remote similarity with the *freeloaders problem*, and to promote altruism that will lead to a superior but unstable equilibrium, one has to introduce enough disincentive against the use of selfish caching.

References

[1] Abraham, I., Awerbuch, B., Azar, Y., et al. *A generic scheme for building overlay networks in adversarial scenarios.* IPDPS 2003.

[2] AODV - mobile ad-hoc paper ... ?

[3] Bejan, A. Ghosh, S. *Self-Optimizing DHT using Request Profiling.* In Proceedings of OPODIS 2004, pp. 87-95 (also in LNCS 3544, pp. 140-153).

[4] Chun B.G., Chaudhuri K., Wee H., et al. *Selfish Caching in Distributed Systems: A Game - Theoretic Analysis.* PODC 2004.

[5] Iamnitchi, A., Ripeanu, M., Foster, I. *Small-World File-Sharing Communities.* IEEE InfoCom 2004, March 2004.

[6] Iamnitchi, A., Ripeanu, M., Foster, I. *Locating Data in Peer-to-Peer Scientific Collaborations.* IPTPS 2002, March 2002.

[7] Kleinberg, J. *The Small-World Phenomenon: An Algorithmic Perspective.* STOC 2000.

[8] Ludwig, R. *A case of Flow-Adaptive Wireless Links.* Tech. Report UCB//CSD-99-1053, University of California at Berkeley, 1999.

[9] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker,S. *A Scalable Content Addressable Network.* ACM SIGCOMM 2001.

[10] Reshef, E. *Approximating Minimum Communication Cost Spanning Trees and Related Problems.* M. Sc. Thesis, Weizmann Institute of Science, 1999.

[11] Rowstron, A. and Druschel, P. *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.* IFIP/ACM International Conference on Distributed Systems Platforms (Middleware) 2001, p. 329-350.

[12] Roughgarden T., and Tardös. Selfish Routing.

[13] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H. *Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications.* IEEE Transactions on Networking 11 (1) 2003.

[14] Triantafillou, P. *Peer-to-Peer Network Architectures: The Next Step (Harnessing the Symbiosis of Altruism and Selfishness).* HDMS 2003.

[15] Zhao, B.Y., Kubiatowicz, J.D., and Joseph, A.D. *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing.* Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.