Routing Algorithms for DHTs: Some Open Questions

Sylvia Ratnasamy
(sylviar@cs.berkeley.edu)

Scott Shenker
(shenker@icsi.berkeley.edu)

Ion Stoica
(istoica@cs.berkeley.edu)

1 Introduction

Even though they were introduced only a few years ago, peer-to-peer (P2P) filesharing systems are now one of the most popular Internet applications and have become a major source of Internet traffic. Thus, it is extremely important that these systems be scalable. Unfortunately, the initial designs for P2P systems have significant scaling problems; for example, Napster has a centralized directory service, and Gnutella employs a flooding-based search mechanism that is not suitable for large systems.

In response to these scaling problems, several research groups have (independently) proposed a new generation of scalable P2P systems that support a *distributed hash table* (DHT) functionality; among them are Tapestry [15], Pastry [6], Chord [14], and Content-Addressable Networks (CAN) [10]. In these systems, which we will call DHTs, files are associated with a key (produced, for instance, by hashing the file name) and each node in the system is responsible for storing a certain range of keys. There is one basic operation in these DHT systems, `lookup(key)`, which returns the identity (*e.g.*, the IP address) of the node storing the object with that key. This operation allows nodes to `put` and `get` files based on their key, thereby supporting the hash-table-like interface.[1]

This DHT functionality has proved to be a useful substrate for large distributed systems; a number of projects are proposing to build Internet-scale facilities layered above DHTs, including distributed file systems [5, 7, 4], application-layer multicast [11, 16], event notification services [3, 1], and chat services [2]. With so many applications being developed in so short a time, we expect the DHT functionality to become an integral part of the future P2P landscape.

The core of these DHT systems is the routing algorithm. The DHT nodes form an overlay network with each node having several other nodes as neighbors. When a `lookup(key)` is issued, the lookup is routed through the overlay network to the node responsible for that key. The scalability of these DHT algorithms is tied directly to the efficiency of their routing algorithms.

Each of the proposed DHT systems listed above – Tapestry, Pastry, Chord, and CAN – employ a different routing algorithm. Usually discussion of DHT routing issues is in the context of one particular algorithm. And, when more than one is mentioned, they are often compared in competitive terms in an effort to determine which is "best". We think both of these trends are wrong. The algorithms have more commonality than differences, and each algorithm embodies some insights about routing in overlay networks. Rather than always working in the context of a single algorithm, or comparing the algorithms competitively, a more appropriate goal would be to combine these insights, and seek new insights, to produce even better algorithms. In that spirit we describe some issues relevant to routing algorithms and identify some open research questions. Of course, our list of questions is not intended to be exhaustive, merely illustrative.

As should be clear by our description, this paper is not about finished work, but instead is about a research agenda for future work (by us and others). We hope that presenting such a discussion to this audience will promote synergy between research groups in this area and help clarify some of the underlying issues. We should note that there are many other interesting issues that remain to be resolved in these DHT systems, such as security and robustness to attacks, system monitoring and maintenance, and indexing and keyword searching. These issues will doubtless be discussed elsewhere in this workshop. Our focus on routing algorithms is not intended to imply that these other issues are of secondary importance.

We first (very) briefly review the routing algorithms used in the various DHT systems in Section 2. We then,

[1] The interfaces of these systems are not all identical; some reveal only the `put` and `get` interface while others reveal the `lookup(key)` function directly. However, the above discussion refers to the underlying functionality and not the details of the API.

in the following sections, discuss various issues relevant to routing: state-efficiency tradeoff, resilience to failures, routing hotspots, geography, and heterogeneity.

2 Review of Existing Algorithms

In this section we review some of the existing routing algorithms. All of them take, as input, a `key` and, in response, route a message to the node responsible for that key. The keys are strings of digits of some length. Nodes have identifiers, taken from the same space as the keys (*i.e.*, same number of digits). Each node maintains a routing table consisting of a small subset of nodes in the system. When a node receives a query for a key for which it is not responsible, the node routes the query to the neighbor node that makes the most "progress" towards resolving the query. The notion of progress differs from algorithm to algorithm, but in general is defined in terms of some distance between the identifier of the current node and the identifier of the queried key.

Plaxton *et al.*: Plaxton *et al.* [9] developed perhaps the first routing algorithm that could be scalably used by DHTs. While not intended for use in P2P systems, because it assumes a relatively static node population, it does provide very efficient routing of lookups. The routing algorithm works by "correcting" a single digit at a time: if node number 36278 received a lookup query with key 36912, which matches the first two digits, then the routing algorithm forwards the query to a node which matches the first three digits (*e.g.*, node 36955). To do this, a node needs to have, as neighbors, nodes that match each prefix of its own identifier but differ in the next digit. For a system of n nodes, each node has on the order of $O(\log n)$ neighbors. Since one digit is corrected each time the query is forwarded, the routing path is at most $O(\log n)$ overlay (or application-level) hops.

This algorithm has the additional property that if the n^2 node-node latencies (or "distances" according to some metric) are known, the routing tables can be chosen to minimize the expected path latency and, moreover, the latency of the overlay path between two nodes is within a constant factor of the latency of the direct underlying network path between them.

Tapestry: Tapestry [15] uses a variant of the Plaxton *et al.* algorithm. The modifications are to ensure that the design, originally intended for static environments, can adapt to a dynamic node population. The modifications are too involved to describe in this short review. However, the algorithm maintains the properties of having $O(\log n)$ neighbors and routing with path lengths of $O(\log n)$ hops.

Pastry: In Pastry [6], nodes are responsible for keys that are the closest numerically (with the keyspace considered as a circle). The neighbors consist of a *Leaf Set* L which is the set of $|L|$ closest nodes (half larger, half smaller). Correct, not necessarily efficient, routing can be achieved with this leaf set. To achieve more efficient routing, Pastry has another set of neighbors spread out in the key space (in a manner we don't describe here). Routing consists of forwarding the query to the neighboring node that has the longest shared prefix with the key (and, in the case of ties, to the node with identifier closest numerically to the key). Pastry has $O(\log n)$ neighbors and routes within $O(\log n)$ hops.

Chord: Chord [14] also uses a one-dimensional circular key space. The node responsible for the key is the node whose identifier most closely follows the key (numerically); that node is called the key's *successor*. Chord maintains two sets of neighbors. Each node has a *successor list* of k nodes that immediately follow it in the key space. Routing correctness is achieved with these lists. Routing efficiency is achieved with the *finger list* of $O(\log n)$ nodes spaced exponentially around the key space. Routing consists of forwarding to the node closest, but not past, the key; pathlengths are $O(\log n)$ hops.

CAN: CAN chooses its keys from a d-dimensional toroidal space. Each node is associated with a hypercubal region of this key space, and its neighbors are the nodes that "own" the contiguous hypercubes. Routing consists of forwarding to a neighbor that is closer to the key. CAN has a different performance profile than the other algorithms; nodes have $O(d)$ neighbors and pathlengths are $O(dn^{\frac{1}{d}})$ hops. Note, however, that when $d = \log n$, CAN has $O(\log n)$ neighbors and $O(\log n)$ pathlengths like the other algorithms.

3 State-Efficiency Tradeoff

The most obvious measure of the efficiency of these routing algorithms is the resulting pathlength. Most of the algorithms have pathlengths of $O(\log n)$ hops, while CAN has longer paths of $O(dn^{\frac{1}{d}})$. The most obvious measure of the overhead associated with keeping routing tables is the number of neighbors. This isn't just a measure of the state required to do routing but it is also a measure of how much state needs to be adjusted when nodes join or leave. Given the prevalence of inexpensive memory and the highly transient user populations in P2P systems, this second issue is likely to be much more important than the first. Most of the algorithms require $O(\log n)$ neighbors, while CAN requires only $O(d)$ neighbors.

Ideally, one would like to combine the best of these two classes of algorithms in hybrid algorithms that achieve short pathlengths with a fixed number of neighbors.

Question 1 *Can one achieve $O(\log n)$ pathlengths (or better) with $O(1)$ neighbors?*

One would expect that, if this were possible, that some other aspects of routing would get worse.

Question 2 *If so, are there other properties (such as those described in the following sections) that are made worse in these hybrid routing algorithms?*

4 Resilience to Failures

The above routing results refer to a perfectly functioning system with all nodes operational. However, P2P nodes are notoriously transient and the resilience of routing to failures is a very important consideration. There are (at least) three different aspects to resilience.

First, one needs to evaluate whether routing can continue to function (and with what efficiency) as nodes fail without any time for other nodes to establish other neighbors to compensate; that is the neighboring nodes know that a node has failed, but they don't establish any new neighbor relations with other nodes. We will call this *static resilience* and measure it in terms of the percentage of reachable key locations and of the resulting average path length.

Question 3 *Can one characterize the static resilience of the various algorithms? What aspects of these algorithms lead to good resilience?*

Second, one can investigate the resilience when nodes have a chance to establish some neighbors, but not all. That is, when nodes have certain "special" neighbors, such as the *successor list* or the *Leaf Set*, and these are re-established after a failure, but no other neighbors are re-established (such as the *finger set*). The presence of these special neighbors allow one to prove the correctness of routing, but the following question remains:

Question 4 *To what extent are the observed path lengths better than the rather pessimistic bounds provided by the presence of these special neighbors?*

Finally, one can ask how long it takes various algorithms to fully recover their routing state, and at what cost (measured, for example, by the number of nodes participating in the recovery or the number of control messages generated for recovery).

Question 5 *How long does it take, on average, to recover complete routing state? And what is the cost of doing so?*

A related question is:

Question 6 *Can one identify design rules that lead to shorter and/or cheaper recoveries?*

For instance, is symmetry (where the node neighbor relation is symmetric) important in restoring state easily? One could also argue that in the face of node failure, having the routing automatically send messages to the correct alternate node (*i.e.* the node that takes over the range of the identifier space that was previously held by the failed node) leads to quicker recovery.

5 Routing Hot Spots

When there is a hotspot in the query pattern, with a certain key being requested extremely often, then the node holding that key may become overloaded. Various caching and replication schemes have been proposed to overcome this *query hotspot* problem; the effectiveness of these schemes may vary between algorithms based on the fan-in at the node and other factors, but this seems to be a manageable problem. More problematic, however, is if a node is overloaded with too much routing traffic. These *routing hotspots* are harder to deal with since there is no local action the node can take to redirect the routing load. Some of the *proximity* techniques we describe below might be used to help here, but otherwise this remains an open problem.

Question 7 *Do routing hotspots exist and, if so, how can one deal with them?*

6 Incorporating Geography

The efficiency measure used above was the number of application-level hops taken on the path. However, the true efficiency measure is the end-to-end latency of the path. Because the nodes could be geographically dispersed, some of these application-level hops could involve transcontinental links, and others merely trips across a LAN; routing algorithms that ignore the latencies of individual hops are likely to result in high-latency paths. While the original "vanilla" versions of some of these routing algorithms did not take these hop latencies into account, almost all of the "full" versions of the algorithms make some attempt to deal with the geographic proximity of nodes. There are (at least) three ways of coping with geography.

Proximity Routing: Proximity routing is when the routing choice is based not just which neighboring node makes the "most" progress towards the key, but is also based on which neighboring node is "closest" in the sense of latency. Various algorithms implement proximity routing differently, but they all adopt the same basic approach of weighing progress in identifier space against cost in latency (or geography). Simulations have shown this to be a very effective tool in reducing the average path latency.

Question 8 *Can one formally characterize the effectiveness of these proximity routing approaches?*

Proximity Neighbor Selection: This is a variant of the idea above, but now the proximity criterion is applied when choosing neighbors, not just when choosing the next hop.

Question 9 *Can one show that proximity neighbor selection is always better than proximity routing? Is this difference significant?*

As mentioned earlier, if the n^2 node-pair distances (as measured by latency) are known, the Plaxton/Tapestry algorithm can choose the neighbors so as to minimize the expected overlay path latency. This is an extremely important property, that is (so far) the exclusive domain of the Plaxton/Tapestry algorithms. We don't whether other algorithms can adopt similar approaches.

Question 10 *If one had the full n^2 distance matrix, could one do optimal neighbor selection in algorithms other than Plaxton/Tapestry?*

Geographic Layout: In most of the algorithms, the node identifiers are chosen randomly (*e.g.* hash functions of the IP address, etc.) and the neighbor relations are established based solely on these node identifiers. One could instead attempt to choose node identifiers in a geographically informed manner.[2] An initial attempt to do so in the context of CAN was reported on in [12]; this approach was quite successful in reducing the latency of paths. There was little in the layout method specific to CAN, but the high-dimensionality of the key space may have played an important role; recent work [8] suggests that latencies in the Internet can be reasonably modeled by a d-dimension geometric space with $d \geq 2$. This raises the question of whether systems that use a one-dimensional key set can adequately mimic the geographic layout of the nodes.

Question 11 *Can one choose identifiers in a one-dimensional key space that will adequately capture the geographic layout of nodes?*

However, this may not matter because the geographic layout may not offer significant advantages over the two proximity methods.

Question 12 *Can the two* local *techniques of proximity routing and proximity neighbor selection achieve most of the benefit of* global *geographic layout?*

Moreover, these geographically-informed layout methods may interfere with the robustness, hotspot, and other properties mentioned in previous sections.

Question 13 *Does geographic layout have an impact on resilience, hotspots, and other aspects of performance?*

7 Extreme Heterogeneity

All of the algorithms start by assuming that all nodes have the same capacity to process messages and then, only later, add on techniques for coping with heterogeneity.[3] However, the heterogeneity observed in current P2P populations [13] is quite extreme, with differences of several orders of magnitude in bandwidth. One can ask whether the routing algorithms, rather than merely *coping* with heterogeneity, should instead use

[2]Note that geographic layout differs from the two above *proximity* methods in that here there is an attempt to affect the global layout of the node identifiers, whereas the proximity methods merely affect the local choices of neighbors and forwarding nodes.

[3]The authors of [13] deserve credit for bringing the issue of heterogeneity to our attention.

it to their *advantage*. At the extreme, a star topology with all queries passing through a single hub node and then routed to their destination would be extremely efficient, but would require a very highly capable nub node (and would have a single point of failure). But perhaps one could use the very highly capable nodes as mini-hubs to improve routing. In another position paper here, some of us argue that heterogeneity can be used to make Gnutella-like systems more scalable. The question is whether one could similarly modify the current DHT routing algorithms to exploit heterogeneity:

Question 14 *Can one redesign these routing algorithms to exploit heterogeneity?*

It may be that no sophisticated modifications are needed to leverage heterogeneity. Perhaps the simplest technique to cope with heterogeneity, one that has already been mentioned in the literature, is to *clone* highly capable nodes so that they could serve as multiple nodes; *i.e.*, a node that was 10 times more powerful than other nodes could function as 10 virtual nodes.[4] When combined with proximity routing and neighbor selection, cloning would allow nodes to route to themselves and thereby "jump" in key space without any forwarding hops.

Question 15 *Does cloning plus proximity routing and neighbor selection lead to significantly improved performance when the node capabilities are extremely heterogeneous?*

References

[1] A. ROWSTRON, A-M. KERMARREC, M. C., AND DRUSCHEL, P. Scribe: The design of a large-scale event notification infrastructure. In *Proceedings of NGC 2001* (Nov. 2001).

[2] BASED CHAT, C. http://jxme.jxta.org/demo.html, 2001.

[3] CABRERA, L. F., JONES, M. B., AND THEIMER, M. Herald: Achieving a global event notification service. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)* (Elmau/Oberbayern, Germany, May 2001).

[4] DABEK, F., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND STOICA, I. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)* (To appear; Banff, Canada, Oct. 2001).

[4]This technique has already been suggested for some of the algorithms, and could easily be applied to the others. However, in some algorithms it would require alteration in the way the node identifiers were chosen so that they weren't tied to the IP address of the node.

[5] DRUSCHEL, P., AND ROWSTRON, A. Past: Persistent and anonymous storage in a peer-to-peer networking environment. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS 2001)* (Elmau/Oberbayern, Germany, May 2001), pp. 65–70.

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

[7] KUBIATOWICZ, J., BINDEL, D., CHEN, Y., CZERWINSKI, S., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. OceanStore: An architecture for global-scale persistent storage. In *Proceeedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)* (Boston, MA, November 2000), pp. 190–201.

[8] NG, E., AND ZHANG, H. Towards global network positioning. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop 2001* (Nov. 2001).

[9] PLAXTON, C., RAJARAMAN, R., AND RICHA, A. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ACM SPAA* (Newport, Rhode Island, June 1997), pp. 311–320.

[10] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *Proc. ACM SIGCOMM* (San Diego, CA, August 2001), pp. 161–172.

[11] RATNASAMY, S., HANDLEY, M., KARP, R., AND SHENKER, S. Application-level Multicast using Content-Addressable Networks. In *Proceedings of NGC 2001* (Nov. 2001).

[12] RATNASAMY, S., HANDLEY, M., RICHARDKARP, AND SHENKER, S. Topologically-aware overlay construction and server selection. In *Proceedings of Infocom '2002* (Mar. 2002).

[13] SAROIU, S., GUMMADI, K., AND GRIBBLE, S. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Conferencing and Networking* (San Jose, Jan. 2002).

[14] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference* (San Diego, California, August 2001).

[15] ZHAO, B. Y., KUBIATOWICZ, J., AND JOSEPH, A. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, University of California at Berkeley, Computer Science Department, 2001.

[16] ZHUANG, S., ZHAO, B., JOSEPH, A. D., KATZ, R. H., AND KUBIATOWICZ, J. Bayeux: An architecture for wide-area, fault-tolerant data dissemination. In *Proceedings of NOSS-DAV'01* (Port Jefferson, NY, June 2001).