1. Topology-aware routing in structured peer-to-peer overlay networks

Miguel Castro[1], Peter Druschel[2], Y. Charlie Hu[3], and Antony Rowstron[1]

[1] Microsoft Research, 7 J J Thomson Avenue, Cambridge, CB3 0FB, UK
[2] Rice University, 6100 Main Street, MS-132, Houston, TX 77005, USA
[3] Purdue University, 1285 EE Building, West Lafayette, IN 47907, USA

1.1 Introduction

Structured peer-to-peer (p2p) overlay networks like CAN, Chord, Pastry and Tapestry [1.14, 1.20, 1.17, 1.22] provide a self-organizing substrate for large-scale p2p applications. They can implement a scalable, fault-tolerant distributed hash table (DHT), in which any item can be located within a small number of routing hops using a small per-node routing table. These systems have been used in a variety of distributed applications, including distributed stores [1.7, 1.18, 1.10, 1.6], event notification, and content distribution [1.23, 1.5, 1.9, 1.4].

It is critical for overlay routing to be aware of the network topology. Otherwise, each routing hop takes a message to a node with a random location in the Internet, which results in high lookup delays and unnecessary wide-area network traffic. While there are algorithmic similarities among each of these systems, an important distinction lies in the approach they take to topology-aware routing. We present a brief comparison of the different approaches that have been proposed [1.16], and give an outlook on future research directions.

1.2 State of the art

In this section, we outline the state of the art in topology-aware routing for structured p2p overlays. We begin with a brief description of four protocols: Pastry, Tapestry, CAN and Chord. In all protocols, nodes and objects are assigned random identifiers (called *nodeIds* and *keys*, respectively) from a large, sparse id space. A *route* primitive forwards a message to the live node that is closest in the id space to the message's key.

In **Pastry**, keys and nodeIds are 128 bits in length and can be thought of as a sequence of digits in base 16. A node's routing table has about $log_{16}N$ rows and 16 columns (N is the number of nodes in the overlay). The entries in row n of the routing table refer to nodes whose nodeIds share the first n digits with the present node's nodeId. The $(n+1)$th nodeId digit of a node in column m of row n equals m. The column in row n corresponding to the value of the $(n+1)$th digit of the local node's nodeId remains empty. At each routing step in Pastry, a node normally forwards the message to a node whose nodeId shares with the key a prefix that is at least one digit longer than the prefix that the key shares with the present node's id. If no such node is known, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node but is numerically closer to the key than the present node's id. Each Pastry node maintains a set of neighboring nodes in the nodeId space (leaf set), both to locate the destination in the final routing hop, and to store replicas of data items for fault tolerance. The expected number of routing hops is less than $log_{16}N$.

Tapestry is very similar to Pastry but differs in its approach to mapping keys to nodes in the sparsely populated id space, and in how it manages replication. In Tapestry, there is no leaf set and neighboring nodes in the namespace are not aware of each other. When a node's routing table does not have an entry for a node that matches a key's nth digit, the message is forwarded to the node in the routing table with the next higher value in the nth digit modulo 2^b. This procedure, called *surrogate routing*, maps keys to a unique live node if the node routing tables are consistent. For fault tolerance, Tapestry inserts replicas of data items using different keys. The expected number of routing hops is $log_{16}N$.

Chord uses a circular 160 bit id space. Unlike Pastry, Chord forwards messages only clockwise in the circular id space. Instead of the prefix-based routing table in Pastry, Chord nodes maintain a *finger table*, consisting of nodeIds and IP addresses of up to 160 other live nodes. The ith entry in the finger table of the node with nodeId n refers to the live node with the smallest nodeId clockwise from $n + 2^{i-1}$. The first entry points to n's successor, and subsequent entries refer to nodes at repeatedly doubling distances from n. Each node also maintains pointers to its predecessor and to its k successors in the id space (the successor list). Similar to Pastry's leaf set, this successor list is used to replicate objects for fault tolerance. The expected number of routing hops in Chord is $\frac{1}{2}log_2N$.

CAN routes messages in a d-dimensional space, where each node maintains a routing table with $O(d)$ entries and any node can be reached in $O(dN^{1/d})$ routing hops. The entries in a node's routing table refer to its neighbors in the d-dimensional space. Unlike Pastry, Tapestry and Chord, CAN's routing table does not grow with the network size but the number of routing hops grows faster than $logN$ in this case, namely $O(dN^{1/d})$.

Next, we describe and compare the three approaches to topology-aware routing in structured overlay networks that have been proposed: *proximity routing*, *topology-based nodeId assignment*, and *proximity neighbor selection* [1.16].

Proximity routing: With proximity routing, the overlay is constructed without regard for the physical network topology. But when routing a message, there are potentially several nodes in the routing table closer to the message's key in the id space. The idea is to select, among this set of possible next hops, the one that is closest in the physical network or one that represents a good compromise between progress in the id space and proximity. With k alternative hops in each step, the approach can reduce the expected delay in each hop from the average delay between two nodes to the average delay to the nearest among k nodes with random locations in the network. The benefits are proportional to the value of k. Increasing k requires a larger routing table with correspondingly higher overheads for maintaining the overlay. Moreover, choosing the lowest delay hop greedily may lead to an increase in the total number of hops taken. While proximity routing can yield significant improvements over a system with no topology-aware routing, its cost/benefit ratio falls short of the other two approaches. The technique has been used in CAN and Chord [1.14, 1.7].

Topology-based nodeId assignment: Topology-based nodeId assignment attempts to map the overlay's logical id space onto the physical network such that neighboring nodes in the id space are close in the physical network. A version of this technique was implemented in CAN [1.14, 1.15]. It achieves a delay stretch (i.e., relative delay to IP) of two or less. However, the approach has several drawbacks. First, it destroys the uniform population of the id space, which causes load balancing problems in the overlay. Second, the approach does not work well in overlays that use a one-dimensional id space (Chord, Tapestry, Pastry) because the mapping is overly constrained. Lastly, neighboring nodes in the id space are more likely to suffer correlated failures, which can have implications for robustness and security in protocols like Chord and Pastry that replicate objects on neighbors in the id space.

Proximity neighbour selection: Like the previous technique, proximity neighbor selection constructs a topology-aware overlay. But instead of biasing the nodeId assignment, the idea is to choose routing table entries to refer to the topologically closest node among all nodes with nodeId in the desired portion of the id space. The success of this technique depends on the degree of freedom an overlay protocol has in choosing routing table entries without affecting the expected number of routing hops. In prefix-based protocols like Tapestry and Pastry, the upper levels of the routing table allow great freedom in this choice, with lower levels having exponentially less choice. As a result, the expected delay of the first hop is very low and it increases exponentially with each hop. Therefore, the delay of the final hop dominates. This leads to low delay stretch, good load balancing, and local route convergence [1.3]. A limitation of this technique is that it does not work for overlay protocols like CAN and Chord, which require that routing table entries refer to specific points in the id space.

Discussion: Proximity routing is the most light-weight technique because it does not construct a topology-aware overlay. But, its performance is limited because it reduces the expected per-hop delay to the expected delay to the nearest among a (usually small) number k of nodes with random locations in the network. Increasing k also increases the overhead of maintaining the overlay. With topology-aware nodeId assignment, the expected per-hop delay can be as low as the average delay among neighboring overlay nodes in the network. However, the technique introduces load imbalance and requires a high-dimensional id space to be effective.

Proximity-neighbor selection can be viewed as a compromise that preserves the load balance and robustness afforded by a random nodeId assignment but still achieves a small constant delay stretch. In a full-length version of this paper [1.3], we show that: proximity neighbor selection can be implemented in Pastry and Tapestry with low overhead; it achieves comparable delay stretch to topology-based nodeId assignment without sacrificing load balancing or robustness; and it has additional route convergence properties that facilitate efficient caching and group communication in the overlay. Moreover, we confirm these results via simulations on two large-scale Internet topology models, and via measurements in a small Internet testbed.

Experience shows that topology-aware routing is critical for application performance. Without it, each routing hop incurs wide-area network traffic and has an average delay equal to the average delay between nodes in the Internet.

1.3 Future research directions

Large-scale simulations using Internet topology models and small-scale Internet testbed experiments show the effectiveness of topology-aware routing in p2p overlays. However, given the complexity of the Internet, larger-scale experiments on the live Internet are necessary to confirm these results and refine the algorithms. In the near term, we expect the PlanetLab effort [1.1] to provide a medium-scale testbed for this purpose. Longer term, it will be necessary to deploy an application that is able to attract a very large user community. It is currently an open question whether proximity neighbor selection can be applied to CAN and Chord, or if equally effective techniques exist that work in CAN and Chord.

Topology-aware routing is currently able to achieve an average delay stretch of 1.4 to 2.2, depending on the Internet topology model. A question is whether this figure can be improved further in a cost-effective manner. One possible approach is to directly exploit Internet topology information from IP or BGP routing tables, or via limited manual configuration. The Brocade effort [1.21] adds hierarchy to Tapestry, where manually configured supernodes route messages among intra-AS Tapestry overlays. However, the performance results are not significantly better than those reported with a flat Pastry overlay [1.3], and supernodes make self-organization, load balance, and security more difficult.

Beyond topology-aware routing, many significant research challenges remain in the area of p2p overlays. In many environments, p2p overlays and the applications built upon them must be tolerant of participating nodes that act maliciously. Initial work has been done in securing the routing and lookup functions [1.19, 1.12, 1.2], and securing application data and services [1.10, 1.13]. However, more effective defenses are needed against the Sybil attack [1.8], where an attacker joins the overlay under many different identities in order to control a fraction of the overlay sufficiently large to compromise security. Moreover, effective solutions must be found to ensure that participating nodes contribute resources proportional to the benefit they derive from the system.

It is also an open question whether structured p2p overlays are suitable in a highly dynamic environment where the set of participating nodes or the underlying physical network topology changes very rapidly. In these environment, unstructured p2p overlays that rely on random search to locate objects may have an advantage [1.11]. One could envision hybrid overlays, where short-term or mobile participants join an unstructured overlay that is connected to a structured overlay consisting of more stable, well-connected participants.

Finally, p2p overlays have been shown to support useful applications like large-scale network storage, event notification, and content distribution. The hope is that they will ultimately prove to enable a larger class of novel, yet to be discovered applications.

10

1.1 Planetlab. http://www.planet-lab.org.
1.2 M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. OSDI'02*, Dec. 2002.
1.3 M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks, 2002. Technical report MSR-TR-2002-82.
1.4 M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: A bandwidth-intensive content streaming system, 2002. Submitted for publication.
1.5 M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8), Oct. 2002.
1.6 L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. In *Proc. OSDI'02*, Dec. 2002.
1.7 F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. SOSP*, Oct. 2001.
1.8 J. Douceur. The Sybil attack. In *Proc. IPTPS'02*, Mar. 2002.
1.9 S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized peer-to-peer web cache. In *Proc. PODC'02*, July 2002.
1.10 J. Kubiatowicz et al. Oceanstore: An architecture for global-scale persistent store. In *Proc. ASPLOS'2000*, Nov. 2000.
1.11 Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make Gnutella scalable? In *Proc. IPTPS'02*, Mar. 2002.
1.12 N. Lynch, D. Malkhi, and D. Ratajczak. Atomic data access in content addressable networks. In *Proc. IPTPS'02*, Mar. 2002.

1.13 A. Muthitacharoen, R. Morris, T. Gil, and B. Chen. Ivy: A read/write peer-to-peer file system. In *OSDI'02*, Dec. 2002.
1.14 S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. SIGCOMM'01*, Aug. 2001.
1.15 S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proc. INFOCOM'02*, 2002.
1.16 S. Ratnasamy, S. Shenker, and I. Stoica. Routing algorithms for DHTs: Some open questions. In *Proc. IPTPS'02*, Mar. 2002.
1.17 A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. Middleware'01*, 2001.
1.18 A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. SOSP*, Oct. 2001.
1.19 E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proc. IPTPS'02*, Mar. 2002.
1.20 I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. SIGCOMM'01*, 2001.
1.21 B. Zhao, Y. Duan, L. Huang, A. Joseph, and J. Kubiatowicz. Brocade: Landmark routing on overlay networks. In *Proc. IPTPS'02*, Mar. 2002.
1.22 B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, Apr. 2001.
1.23 S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination. In *Proc. NOSSDAV'01*, June 2001.