Consistent Key Mapping in Structured Overlays

Andreas Haeberlen Jeff Hoye Alan Mislove Peter Druschel

Rice University, Houston, TX, USA
{ahae,jeffh,amislove,druschel}@cs.rice.edu

Abstract

Most structured peer-to-peer overlays rely on consistent hashing to determine the node that is responsible for a given key. For consistent hashing to work properly, it requires that the nodes have a consistent view of their neighborhood in the identifier space. However, if routing anomalies occur in the underlying network, this view can become inconsistent, causing unstable overlay behavior and, worse, allowing more than one node to assume responsibility for ranges of keys.

We present a set of techniques for preventing inconsistencies under routing anomalies, and we propose to adopt strategies from mobile ad-hoc networking for maintaining connectivity in the presence of path failures. We evaluate our design in the context of Pastry and present results from a deployment in the PlanetLab testbed.

1 Introduction

Peer-to-peer (p2p) overlays are usually built on the assumption that the underlying network provides full connectivity. Unfortunately, experience has shown that this assumption is too optimistic. Real networks suffer from a variety of routing problems, such as loops, misconfigurations, route 'fluttering', and infrastructure failures [13, 16, 18]. Almost all of these anomalies are temporary, but when they do occur, some unlucky nodes may lose connectivity to part of the network. Many anomalies are not even symmetric; thus, messages from node *A* may still reach node *B* when, from *B*'s point of view, *A* has already become unreachable.

In general, the presence of path failures has little effect on overlay networks, since they can use alternate routes – systems like RON [1] directly exploit this to improve routing performance. However, path failures may affect the nodes' perception of overlay membership. For example, if a path between two overlay members *A* and *B* becomes unavailable, *B* may be led to believe that *A* has left the overlay, and vice versa.

In structured overlays like Pastry [17] and Chord [19], where information about nearby nodes is used to implement consistent hashing, such inconsistent views can have serious effects, since they may cause two nodes to assume responsibility for the same part of the key space. As a consequence, applications built on top of the overlay may violate mutual exclusion, suffer data loss, or experience diverging state. For example, an overlay-based file system might use consistent hashing to associate each file with a single node to serialize updates. In this case, inconsistencies can cause multiple nodes to accept updates for the same file. Thus, some updates may be applied incompletely, in the wrong order, or not at all.

The techniques used in most overlays to resolve inconsistencies are not designed for an environment with path failures; in fact, some explicitly make the assumption that path failures do not occur [4]. However, using results from a recent study of connectivity in the PlanetLab testbed, we found that path failures are a common phenomenon in the Internet today. During a period of 10 days in September 2004, all of the 192 nodes we examined experienced at least one path failure. While 35% of the failures lasted less than one hour, 9% of the failures persisted for more than a day. This result, which is consistent with other studies [6,12,13], clearly demonstrates that overlay protocols must be designed and evaluated in an environment in which path failures occur.

In this paper, we present a set of techniques to prevent such inconsistencies by ensuring that at any time, at most one node is responsible for any given key. Our failure model explicitly includes asymmetric connectivity and network partitions. We adopt techniques from routing in mobile ad-hoc networks, specifically from the DSR [11] protocol, to maintain connectivity in the presence of path failures. As an additional benefit, our solution naturally allows nodes behind NATs and firewalls to participate in the system.

We have already integrated some of our techniques with the FreePastry [9] implementation of Pastry, and we demonstrate their effectivity by reporting preliminary results from a PlanetLab deployment.

The rest of this paper is structured as follows: Section 2 discusses related work, and Section 3 presents results from our study of routing anomalies in Planet-

Lab. In Section 4, we describe the design of our resilient transport layer and argue for its correctness. Section 5 describes our experience with the redesigned Pastry protocol and Section 6 presents our conclusions.

2 Related Work

RON [1] is an overlay that is explicitly designed to optimize network performance in the presence of path failures. In a RON overlay, traffic can be re-routed around a failure via a multi-hop virtual link. However, since RON provides essentially a best-effort service, it does not have a strong consistency requirement like Pastry and thus can use a much simpler membership protocol.

UIP [8] uses virtual links to form connections between nodes in arbitrary topologies, which may include NATs and firewalls. However, UIP provides only basic connectivity and no higher-level primitives such as consistent hashing, so consistency is not an issue.

Bamboo [15] is a variant of Pastry that has extensions for better performance under high churn. The Bamboo paper was the first quantitative study of routing inconsistencies in Pastry, although the authors considered only the impact of churn and not that of path failures. In networks such as PlanetLab, where path failures are common, Bamboo still offers high stability; however, it cannot guarantee consistency.

Castro et al. [4] describe a set of extensions to MSPastry which, among other things, explicitly address the issue of routing consistency under churn. However, the authors a) use direct probing to detect failures, and b) assume that non-faulty nodes are never considered faulty. Path failures violate this assumption and may lead to routing inconsistencies.

In addition to the classical study conducted by Paxson [13], there are several other studies which support our claim that path failures are a common problem. Labovitz et al. studied routing table logs at Internet backbones and found that 10% of all considered routes were available less than 95% of the time [12]. Chandra et al. found that 5% of all detected failures lasted more than 10,000 seconds; some failures persisted for over a day before they were repaired [6].

3 Routing anomalies

Since our initial design assumed full network connectivity, the failure of a TCP connection attempt to a remote node was interpreted as an indication that remote node had failed. This assumption held for the most part within the Riceintranet. However, when we deployed ePOSTon PlanetLab, we observed frequent *routing anomalies*, or instances where live nodes were unable to route packets to each other. Our discovery of routing anomalies is not unique; others have noted similar behavior on the general Internet [2, 10, 14].

Routing anomalies can lead to inconsistencies in overlay operation. Overlay networks such as Pastry tolerate connectivity problems among pairs of nodes: for routing table entries, two nodes that can no longer reach each other simply select alternate entries. As a result, key lookup operations tend to not be affected by routing anomalies, because the overlay generally finds a working overlay route to the node that is responsible for the key. However, following a key lookup, the application usually wishes to open a direct connection between client node and the responsible node, for instance, to efficiently transfer data. If the Internet connection between these two nodes does not work, then the application will observe an anomaly: the responsible node for the key can be looked up and reached via an overlay route, but an attempt to directly connect to that node fails.

A simple workaround would be to fall back on using overlay routes when such a problem occurs. However, we found that this approach tends to congest overlay links with bulk data traffic, which can cause long delays for key lookups throughout the overlay. Before we describe our redesign to address this problem, we first present some data on the frequency of routing anomalies.

3.1 Anomaly frequency

We examined data collected from the PlanetLab testbed collected over 10 days in September of 2004. At the time, PlanetLab consisted of 435 nodes spread over 201 sites, including nodes in both of the Americas, Europe, the Middle East, Asia, and Australia. The data we examined consisted of node-to-node pings collected every 15 minutes over the course of the run, taken from [20].

In order to investigate the impact of routing anomalies, we limited our evaluation to nodes which were online and reachable by at least one other node. This left us with 192 distinct nodes. Figure 1 shows the results of site-to-site pings for the 192 considered nodes. In the left graph, the pixel at the location (x,y) represents the number of times a node x was able to successfully ping node y. White pixels indicate no failures, while darker pixels indicate an increasing frequency of ping failures. The right graph shows pairs that were never successful in pinging each other. The data we used for both graphs was collected between September 1 and September 10, 2004.

These results clearly show that routing anomalies are a persistent problem in PlanetLab. All of the 192 nodes we examined experienced at least one routing anomaly during the experiment; many of them experienced several. The average duration of an anomaly is 8.8 hours,

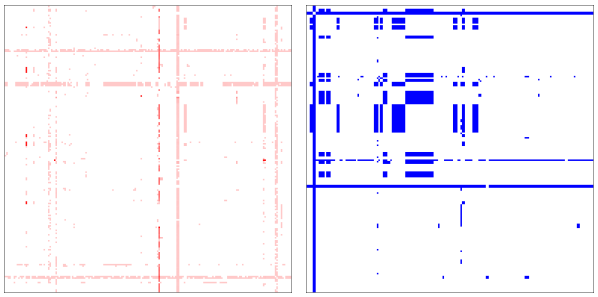

Figure 1. Transient (left) and permanent (right) path failures in PlanetLab during 10 days in September 2004

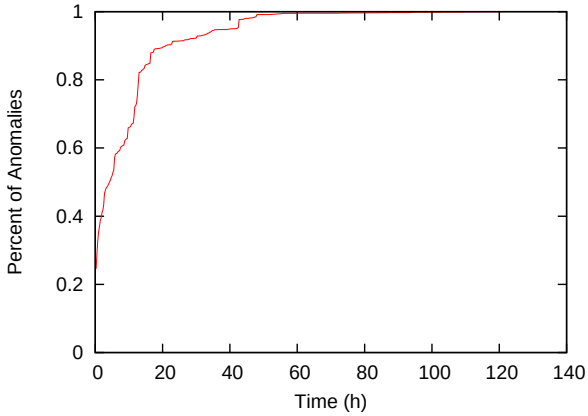

Figure 2. CDF of routing anomaly durations for 192 PlanetLab nodes for September 1 to September 10, 2004.

but the distribution is heavy-tailed; a full 35 % of the outages were present for less than one hour, while 9 % were longer than a day. Figure 2 shows the cumulative distribution of anomaly duration.

3.2 Redesign: Virtual links

To ensure that all pairs of overlay nodes can efficiently communicate despite routing anomalies, we provide dedicated bulk transfer connections via intermediate nodes. We view all node connections as *virtual links*, rather than physical links. For example, assume that a link from $A \rightarrow B$ has failed. If there is another node C that can still reach B, A can replace its direct link $A \rightarrow B$ with a virtual link $A \rightarrow C \rightarrow B$. Note that in the common case where no path failure has occurred, the virtual link is identical to the actual physical link and thus requires no extra overhead. The use of virtual links is a well-known technique that is widely used in mobile ad-hoc networks [11], where path failures are common. On the general Internet virtual links have been used by several systems to route around an increasing number of observed routing anomalies [2, 10].

3.3 Redesign: Source routes

We use *source routing* to forward packets over virtual links. Messages sent via source routes are not subject to normal overlay routing—they are either transmitted along the specified path or dropped if an error occurs. This is important to prevent routing loops, as source routing may not observe the normal Pastry routing invariants (e.g. always routing to a node with nodeId closer to the key). Moreover, such source routed virtual links use dedicated, coupled TCP connections to ensure flow control along the entire virtual link.

Every node periodically advertises its best virtual links to all other leafset members, who use them to derive virtual links for themselves. For example, if A advertises a link $A \rightarrow B$ to C, and C's best link to A is currently $C \rightarrow D \rightarrow A$, then C concludes that B may be reached via $C \rightarrow D \rightarrow A \rightarrow B$. Nodes maintain a set of fresh links for each destination, but during normal operation, only the shortest virtual link is used.

If the shortest virtual link to a destination X is not a direct link, the node occasionally sends a probe packet directly to X, using exponential back-off. If the probe is answered, the physical link to X is re-enabled, and all other virtual links are updated accordingly. This ensures that after a transient path failure, the system eventually returns to using physical links.

When a virtual link fails, the sender starts using another link, if one is available. If not, the sender can broadcast a *route request* to all of its leafset members, who attempt to forward it to the destination. The destination responds with a *route reply*. This mechanism was inspired by the DSR ad hoc routing protocol [11] and does not require connectivity to be symmetric. Thus, it can not only handle asymmetric path failures, it also allows us to incorporate nodes behind NATs, firewalls, and advanced traffic shapers into the overlay, whose connectivity may be asymmetric as well.

3.4 Redesign: Improvement

We incorporated virtual links and source routing into our ePOSTimplementation. As a result, routing anomalies are no longer visible to applications. In a deployment in PlanetLab, the redesigned system communicated between 59,104 distinct pairs of machines and found that 9.1% of these pairs were unable to establish a direct connection. Among these cases that required an indirect route, 9.8% required a source route with just one extra hop. We monitored all of the routes in the network for a period of three days, and found that the percentage of indirect routes varied between 8% and 11%. The results of this experiment are shown in Figure 3. Source routes and virtual links allowed the system to route around network anomalies in all cases, as expected.

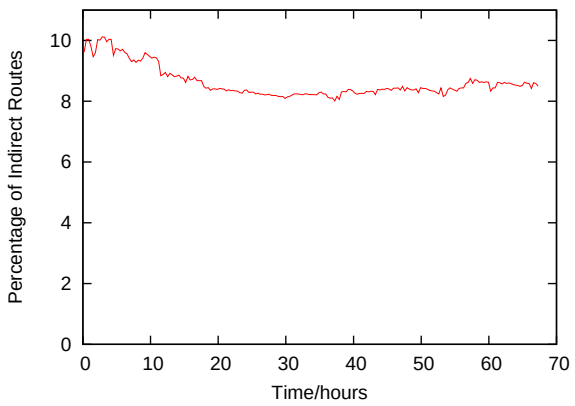

Figure 3. Percentage of indirect routes over a period of three days, March 17 to March 20, 2005.

4 Overlay consistency

In a structured overlay, the responsibilities of a node are largely determined by the region of key space it is currently responsible for. Many subsystems of ePOSTuse this key space information; for example, PAST uses it to determine which objects it should store a replica of, and Scribe uses it to decide whether or not it should be the root of a given multicast tree. Generally, a node will accept a message if its key falls into the range the node is currently responsible for; otherwise it will drop the message or route it to another node.

In the initial design of Pastry, a simple mechanism was used to determine the responsible key range. Nodes would monitor the liveness of nodes in their own leafsets by pinging them occasionally, and remove them if they did not respond for some time. The responsible key range was calculated simply by considering the nodeIds of the two neighboring nodes in the identifier space.

This sufficed within our local ring at Rice. However, when we deployed ePOSTon PlanetLab, we observed severe malfunctions: e-mails would not get delivered, and objects could not be inserted into PAST or would spontaneously roll back to previous versions. The underlying cause was that messages with the same key were delivered to different nodes, depending on the node from which the lookup started. Some investigation revealed that path failures and node overload were causing inconsistencies in the perceived liveness of nodes. This in turn led to inconsistencies between the nodes' leafsets and thus caused messages to be delivered incorrectly. In fact, we examined the running ePOSTnetwork and found that, without virtual links, approximately 5.2% of all routed messages could be claimed by more than one node, mainly due to the large numbers of path failures.

Routing consistency is a fundamental property for a p2p system that needs to maintain mutable data. Without it, replicas of mutable data diverge, updates may be lost and the state of a mutable object may appear inconsistent depending on which replica is reached during a lookup. Other overlays such as Bamboo [3] endeavor to reduce overlay maintenance overhead, but do not attempt to provide routing consistency in the event of path failures. Thus, we decided to redesign Pastry's leafset stabilization protocol to guarantee, with high probability, that only one node considers itself responsible for a given time at any time despite routing anomalies. A similar approach was taken in MSPastry [5], but routing anomalies were not considered as part of that work.

4.1 Redesign: Requirements

Our new leafset stabilization protocol provides routing consistency, which is defined as follows:

With high probability, there is at most one node at any given time t that will accept messages for a key k.

Since the decision whether or not to accept a given message is based on the leafset, the leafset stabilization protocol must satisfy the following constraints:

1. When a new node joins, it must make sure that the current members stop accepting message for their part of the key space *before* the new node starts accepting messages.

2. When a node fails or leaves, it must make sure that the neighbors take over its portion of key space only *after* they have established that the node no longer accepts messages.

3. After joins or departures subside, it must ensure that the leafsets eventually reflect a consistent view of overlay membership.

Our stabilization protocol is based on the assumption that leafsets are always *connected*, i.e. that for each pair of leafset members A and B, there exists a path of leafset nodes A, N_1, ... N_k, B (where N_i is in the leafset) such that each is directly connected to the next, and A is therefore able to send messages to B along this path. This assumption requires that the source routing mechanism described in Section 3.3 discovers a route if one exists. As we will show in Section 4.5, this is a reasonable assumption.

4.2 Liveness checks

Direct neighbors in the key space constantly monitor each other's liveness. For this purpose, they make sure that they receive at least one message from each other within a configurable time period T_P, on the order of 30

seconds. When there is no overlay traffic to send, they may send a `Ping` message instead. Each `Ping` must be answered immediately by a `Pong`, which also must include the source route used in the corresponding `Ping` as payload.

Should a node A not receive any traffic from B after T_P, A starts sending periodic `Pings` to B over the best known virtual link. Should A still not receive any messages after time $2T_P$, A starts sending `Ping` to B via the best known route to each of its leafset members. If a `Pong` arrives now, A changes B's virtual link to the source route listed in the `Pong`.

However, if $3T_P$ expires before any route is found, A has established that *none* of the other leafset members can directly reach B any more, so under our assumption that leafsets are always connected, B cannot be alive. Hence, A declares B dead. However, A does not remove B from its leafset until another T_P has passed, to ensure that all other leafset nodes declare B dead. Once this total of $4T_P$ has expired, A can remove B from the leafset.

The parameter T_P directly influences the bandwidth required for maintenance. Higher values result in lower bandwidth, but also increase the latency between a node failure and the time when its portion of key space is taken over by the other nodes.

4.3 Key ownership

To prevent overlaps between the responsible region of a newly joined node and that of its neighbors, we introduce the concept of *key ownership*. Each node has a range of keys which it owns, and it is not allowed to accept messages for keys outside of this range. When the first node N_1 with nodeId k_1 in the network starts up, it automatically has ownership over the entire key range. As nodes join, they request range transfers from existing nodes. For example, when the next node N_2 with nodeId k_2 boots up, it will ask N_1 to transfer ownership of the range

$$\left[\frac{k_1+k_2}{2} \bmod 2^k, \frac{k_1+k_2+2^{160}}{2} \bmod 2^{160}\right)$$

Note that once a node releases ownership over a range of keys, it is no longer able to accept message for those keys. Additionally, each node must obtain ownership transfers from *both* of its neighbors before accepting any messages. During the interim time period message may dropped at the expense of preserving consistency.

If we assume for the moment that no nodes ever leave the overlay (i.e. all nodes stay forever), it is clear that routing consistency is maintained. Since the key space is repeatedly partitioned up, nodes never conflict in their owned ranges. However, as churn is common and routing anomalies are possible, we must show that reclaiming transferred key space does not break routing consistency. In order to do so, we impose the rule that a node may reclaim its neighbor's owned keys only if the node is declared dead (as discussed in Section 4.2). If a neighbor is declared dead, then, by the assumption that leafsets are connected, we know that no leafset member is able to reach the neighbor, and we can therefore assert that the neighbor is dead. In this case, the remaining node may reclaim its portion of the neighbor's key space.

4.4 Ring partitions

While node failures are handled by the techniques above, a little more is needed to support ring partitions, or instances where a leafset is split into multiple subsets of connected nodes. To maintain consistency under these scenarios, we impose the rule that if a node notices that more than half of its leafset dies within one period ($4T_P$), the node automatically resigns from the network. This makes it likely that, should a network partition occur, only a majority subset, if any, survives. Nodes that resign attempt to rejoin the network, using exponential backoff on retry intervals.

4.5 Justification

When designing this protocol, we assumed that leafsets were connected, or that in a given node N's leafset $[N_1, N_2, ..., N, ..., N_k]$ there always existed a path between N and every N_i. In this section, we provide a quick justification of why this is a reasonable assumption.

As mentioned earlier, we assume that each path $N_i \rightarrow N_j$ in a given leafset fails independently with probability p. For simplicity, we consider virtual links with at most two hops. Then N cannot reach N_i if the direct path $N \rightarrow N_i$ fails *and* for every leafset member L_i, either the path $N \rightarrow L_i$ or the path $L_i \rightarrow N_i$ fails. If the leafset contains l nodes on each side, this occurs with probability

$$P_1 = p \cdot \left(1-(1-p)^2\right)^m$$

where m is the number of nodes in the shared leafset of N and N_i, which ranges from $l-1$, when N and N_i are far apart, to $2(l-1)$, when they are adjacent. We consider N and N_i disconnected if either of them cannot reach the other one. This probability is

$$P_2 = 1-(1-P_1)^2$$

As stated above, a routing consistency is broken only if N is disconnected from either his left or his right neighbor. This happens with probability

$$P_3 = 1-(1-P_2)^2$$

If we assume small leafsets ($l = 8$) and a massive failure of $p = 0.1$, then $P_3 \approx 6.072 \cdot 10^{-12}$, so even in a network with $N = 10000$ nodes, the probability of finding a single disconnected node is less than $6.1 \cdot 10^{-8}$. If we consider virtual links with more than two hops, the resulting probability is even lower. For comparison, in a protocol that requires a physical link between each node and his right and left neighbors, the probability of an inconsistency is

$$P_3' = 1 - (1-p)^4$$

For the parameters mentioned earlier, $P_3' \approx 0.3439$, so about one-third of the nodes would be disconnected.

5 Experience

We have implemented the above techniques into FreePastry [9], and have deployed the implementation on a ring of 320 PlanetLab nodes. While the previous versions of FreePastry suffered from numerous routing inconsistencies when deployed on this set of machines, the new version has been been successfully run for multiple days without any detected routing inconsistencies. Out of the 44,480 detected routes, we found that multiple-hop routes were required in 1,307, or 2.9%, of the cases. The vast majority of these (1,293) were two-hop routes, while three-hop routes were used 13 times and one four-hop route was required.

Additionally, we found the bandwidth overhead of our techniques to be very small - on average, nodes used less than 1 KB/s of bandwidth. Even during the booting phase, where all machines were brought online within 30 minutes and most routes were discovered, the peak bandwidth at any node was under 10 KB/s. This implementation of our protocol will be part of the upcoming FreePastry 1.4 release [9].

6 Conclusions and Future Work

In this paper, we have argued that overlay maintenance protocols must be designed for an environment in which path failures are common. Using experimental data from the PlanetLab testbed, we have demonstrated that long-lived path failures occur frequently in the Internet today, and we have shown that this can lead to routing inconsistencies in overlays, with catastrophic effects on applications. Finally, we have presented the design of a maintenance protocol for the Pastry overlay that increases its resilience against path failures by several orders of magnitude.

References

[1] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, Oct 2001.

[2] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of SOSP'01*, Oct. 2001.

[3] Bamboo DHT. `http://bamboo-dht.org/`.

[4] M. Castro, M. Costa, and A. Rowstron. Performance and dependability of structured peer-to-peer overlays. In *Proceedings of the International Conference on Dependable Systems and Networks*, June 2004.

[5] M. Castro, M. Costa, and A. Rowstron. Performance and dependability of structured peer-to-peer overlays. In *Proceedings of DSN'04*, Florence, Italy, June 2004.

[6] B. Chandra, M. Dahlin, L. Gao, and A. Nayate. End-to-end WAN service availability. In *3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, Mar 2001.

[7] N. Feamster, D. Andersen, H. Balakrishnan, and F. Kaashoek. Measuring the effects of internet path faults on reactive routing, Jun 2003.

[8] B. Ford. Unmanaged internet protocol: Taming the edge network management crisis. In *Proceedings of the 2nd Workshop on Hot Topics in Networks*, Nov 2003.

[9] The FreePastry web site. http://freepastry.rice.edu/.

[10] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, and H. M. Levy. Improving the reliability of internet paths with one-hop source routing. In *Proceedings of OSDI '04*, Dec. 2004.

[11] D. B. Johnson, D. A. Maltz, and J. Broch. The dynamic source routing protocol for multihop wireless ad hoc networks. In *Ad Hoc Networking*, edited by Charles E. Perkins, Chapter 5, pages 139–172. Addison-Wesley, 2001.

[12] C. Labovitz, G. R. Malan, and F. Jahanian. Internet routing instability. In *Proceedings of ACM SIGCOMM*, Sep 1997.

[13] V. Paxson. End-to-end routing behavior in the internet. In *Proceedings of ACM SIGCOMM '96*, Aug 1996.

[14] V. Paxson. End-to-end Internet packet dynamics. In *Proceedings SIGCOMM'97*, Cannes, France, September 1997.

[15] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, June 2004.

[16] M. Roughan, T. Griffin, Z. M. Mao, A. Greenberg, and B. Freeman. Ip forwarding anomalies and improving their detection using multiple data sources. In *Proceedings of the ACM SIGCOMM workshop on Network troubleshooting*, pages 289–294, 2004.

[17] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, Nov 2001.

[18] Z. Shu and Y. Kadobayashi. Troubleshooting on intra-domain routing instability, 2004.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, San Diego, CA, August 2001.

[20] J. Stribling. All pairs ping results for PlanetLab. http://www.pdos.lcs.mit.edu/ strib/pl_app/.